SAVONIA

Verkkopalvelualustat

Tuntikirjaus verkkopalveluna ja Android-sovelluksena

Kimmo Toivanen

Opinnäytetyö

___ . ___ . _____ _____

Ylempi ammattikorkeakoulututkinto

SAVONIA-AMMATTIKORKEAKOULU **OPINNÄYTETYÖ**
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala			
Koulutusohjelma Hyvinvointiteknologian koulutusohjelma			
Työn tekijä(t) Kimmo Toivanen			
Työn nimi Verkkopalvelualustat – tuntikirjaus verkkopalveluna ja Android-sovelluksena			
Päiväys	29.5.2012	Sivumäärä/Liitteet	82/1
Ohjaaja(t) lehtori Jussi Koistinen, lehtori Sami Lahti			
Toimeksiantaja/Yhteistyökumppani(t) SoftRain Blobs Oy			

Tiivistelmä

SoftRain Blobs Oy on ohjelmistoalan yritys, jolla on vahva osaaminen mobiililaitteiden ja automaatiojärjestelmien sekä sulautettujen laitteiden ohjelmistojen suunnittelusta. Nyt yritys hakee uusia mahdollisuuksia verkkopalveluiden suunnittelusta.

Opinnäytetyön tehtävänä oli luoda yritykselle aluksi omaan käyttöön ja myöhemmin asiakkaille palveluna tai ohjelmistona tarjottava tuntikirjaussovellus sekä siihen liittyvä mobiiliohjelma. Samalla tutkittiin verkkopalvelualustoja, joita yritys voi käyttää tulevaisuudessa.

Tässä työssä esitellään lähemmin kolme verkkopalvelualustaa (Spring, ASP.net MVC ja Grails). Menetelminä käytettiin tietojen hakemista kirjallisuudesta ja internetistä sekä verkkopalvelualustojen käytön arviointia. Lisäksi kuvataan tarkemmin joitakin tuntikirjaussovelluksen ominaisuuksia ja toteutustapoja sekä käydään läpi mobiilisovelluksen periaatteita sekä toimintoja.

Tuloksena saatiin yrityksen käyttöön sopiva tuntikirjaussovellus ja Android-sovellus. Verkkopalvelualustoista kaikki sopivat yrityksen käyttöön, vaikkakin Grailsin sijaan kannattanee panostaa johonkin PHP-pohjaiseen verkkopalvelualustaan.

Avainsanat
verkkopalvelu, palvelualusta, mvc, spring, roo, grails, tuntikirjaus, android

SAVONIA UNIVERSITY OF APPLIED SCIENCES **THESIS**
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Welfare Engineering			
Author(s) Kimmo Toivanen			
Title of Thesis Web Frameworks – Hour Reporting as Web Application and Android Application			
Date	29 May 2012	Pages/Appendices	82/1
Supervisor(s) Mr Jussi Koistinen, Lecturer; Mr Sami Lahti, Lecturer			
Client Organisation/Partners SoftRain Blobs Oy			

Abstract

SoftRain Blobs Oy is a software design company with strong know-how in mobile, automation and embedded devices. The company is now looking for new business opportunities in web applications.

The purpose of this work was to create a web application and Android application for hour reporting. The applications were intended for the company's internal use but later could be offered to customers either as a service or as an application. Some web application frameworks were studied to help the company to select frameworks that will be used in the future.

Three separate web application frameworks (Spring, ASP.net MVC and Grails) were studied in greater detail. The methods of the study were searching information from literature and Internet and evaluation of the frameworks. Also some properties and their implementation in a web application and Android application were studied.

The result of this thesis was an hour reporting web application and Android application for the company's internal use. All studied web application frameworks were suitable for the company, though some PHP based web frameworks could be selected instead of Grails.

Keywords
web service, web platform, mvc, spring, roo, grails, hour reporting, android

SISÄLTÖ

1 JOHDANTO

Verkkopalvelu voi olla vaikea erottaa verkkosivuista tai kotisivuista. Verkkosivut ovat kuitenkin yksittäisiä sivuja, sisällöltään enimmäkseen staattisia ja yhdistyvät toisiin verkkosivuihin hyperlinkeillä. Niitä voi vaivattomasti kirjoittaa tavallisella tekstieditorilla. Kotisivuja voi julkaista vaikkapa WordPress- tai Joomla-alustoilla, joilla hallitaan sisältöä. Sisältö tallennetaan esimerkiksi tietokantaan ja yhdistetään osaksi käyttäjälle esitettävää näkymää. Kotisivuissa voi olla jonkin verran dynaamisia elementtejä. Verkkopalvelu on ”*asiakas-palvelin -sovellus, joka käyttää verkkoselainta asiakasohjelmana ja suorittaa interaktiivisia palveluita yhdistymällä palvelimiin internetin tai intranetin kautta*” (Shklar & Rosen 2003).

1.1 Yrityksen toimiala ja kehitystarpeet

SoftRain Blobs Oy on vuonna 2007 perustettu teollisuuden ja elinkeinoelämän ohjelmistoratkaisuja kehittävä yritys. Toiminta alkoi osana Protacon-konsernia ja vuoden 2011 alkupuolelta lähtien SoftRain on toiminut itsenäisenä yrityksenä, ja toimintatapoja ja liikeideoita on jouduttu miettimään uudelleen.

Yksi toimintatavoista on henkilöstön tuntikirjanpito. Vaikka tuntikirjausohjelmistoja on markkinoilla saatavana runsaasti, ei yrityksen tarpeisiin sopivaa ohjelmistoa ole löytynyt. Yrityksellä on henkilöstöä eri puolilla Suomea ja kokemusta mobiilisovellusten kehittämisestä, joten toteutus omasta tuntikirjausohjelmistosta, jota käytetään yrityksen sisäisesti ja mahdollisesti tarjotaan palveluna muille yrityksille, alkoi tuntua sopivimmalta vaihtoehdolta.

Tuntikirjausohjelmisto suunnitellaan sopivaksi pienten ja mahdollisesti keskisuurten yritysten tarpeisiin. Tuntikirjaukseen kukin työntekijä merkitsee esimerkiksi puolen tunnin jaolla päivän aikana tehdyt tehtävät. Tehtävät tallennetaan tietokantaan, josta voidaan ottaa erilaisia raportteja sekä tuntilaskelmia. Suunnitelmissa on myös tulostaa tuntiraportti sellaisessa muodossa, että sen voi syöttää suoraan kirjanpitotoimiston käyttämään ohjelmaan. (Kaukonen & Ronkainen 2011.)
Kaikilla yrityksen työntekijöillä on usean vuoden kokemus sulautettujen ja mobiililaitteiden kehittämisestä, useilla automaatiolaitteiden kehittämisestä sekä joillain mobiilisovellusten kehittämisestä. Yrityksessä ei kuitenkaan vielä ole vahvaa osaamista verkkopalvelujen kehittämisestä.

1.2 Opinnäytetyö

Opinnäytetyön aiheena on Laura Rosenbergin kanssa suunniteltu ja toteutettu tuntikirjaussovellus -verkkopalvelu sekä siihen liittyvä mobiilisovellus. Aiheen toteutus ja raportointi on jaettu tekijöiden kesken. Verkkopalvelulla tarkoitetaan tässä sitä, että sovellusta voidaan käyttää missä tahansa internetselaimella. Itse sovellus toimii verkkopalvelimessa, jonka varsinaisella sijoituspaikalla ei ole sovelluksen käytön kannalta merkitystä. Tuntikirjaussovelluksen toimintoja ovat työntekijän työtuntien kirjaaminen ja siihen liittyvä hallinnointi, kuten projektien luominen ja liittäminen työntekijöihin ja tuntikirjausten hyväksyminen. Myöhemmässä vaiheessa tuntikirjaussovellukseen lisätään matka- ja kululaskujen kirjaaminen ja tilitoimistolle palkanlaskentaa varten lähetettävät raportit.

Tuntikirjaussovellusta voi käyttää tablet-laitteiden ja älypuhelinten selaimella, mutta tämän lisäksi haluttiin tehdä myös erilliset mobiilisovellukset. Mobiilisovellus käyttää tuntikirjaussovellusta tuntikirjausten hakemiseen ja tallentamiseen internetyhteyden kautta. Mobiilisovelluksen etuna on kirjausten tallentaminen laitteeseen silloin, kun internetyhteyttä ei ole saatavilla tai sen käyttäminen ei vaikkapa hinnan takia ole järkevää.

Tässä opinnäytetyössä kuvataan osa tuntikirjaussovelluksen toteutuksesta ja Android-laitteille toteutettu mobiilisovellus, kun taas Rosenberg kuvaa omassa opinnäytetyössään osan tuntikirjaussovelluksen osista ja Windows Phonelle toteutetun mobiilisovelluksen (Rosenberg 2012). Lisäksi tässä opinnäytetyössä tutkitaan kevyesti kolmea erilaista verkkopalvelualustaa, järjestelmää, joilla erilaisia verkkopalveluita tai sovelluksia voidaan luoda.

Tuntikirjaussovellus ja Android-sovellus soveltuvat opinnäytetyön laajuudessa yrityksen sisäiseen käyttöön. Kaupallinen käyttö vaatii lisäkehitystä.

2 VERKKOPALVELUALUSTAT

Verkkopalvelu toimii aina jonkin alustan pohjalta. Alusta voi yksinkertaisimmillaan olla verkkopalvelin, joka osaa tulkita asiakasohjelman pyyntöjä ja vastata niihin toteuttaen palvelun. Käytännössä uutta verkkopalvelualustaa ei kannata ryhtyä rakentamaan (ellei siinä ole jotain poikkeuksellista toiminnan tai oppimisen kannalta), vaan verkkopalvelut pohjautuvat useimmiten johonkin valmiiseen verkkopalvelualustaan. Todennäköisesti verkkopalvelualustan kehittäjät ovat jo tehneet kaikki verkkopalvelun hankalat osat ja jättäneet varsinaisen sovelluksen kehittäjän tehtäväksi.

2.1 Tutkimuksen tavoite

Tutkimuksen tavoitteena on vertailla erilaisten yleisten ja lisenssimaksuttomien verkkopalvelualustojen sopivuutta yrityksen käyttöön. Koska yritys toimii asiakkaittensa ehdoilla, yritys voi ehdottaa asiakkaalle sovelluksessa käytettävää alustaa, mutta asiakas lopulta päättää, millä tavalla sovellus toteutetaan. Tavoitteena on selvittää, millä tavoin kullakin sovellusalustalla voidaan luoda verkkopalveluja, jotta yrityksessä voidaan perustella eri alustojen sopivuutta erilaisiin sovelluksiin ja asiakkaan entisiin järjestelmiin.

Koska yritys on varsin nuori verkkopalveluiden kehittämistyössä, on tämä tutkimus vasta alku valittujen ja muiden alustojen kokeilussa, opiskelussa ja käytössä.

2.2 Tutkimusmenetelmät

Tutkimusmenetelminä ovat alustojen arviointi sekä niiden ominaisuuksien hakeminen kirjallisuudesta ja internetsivuilta. Arvioinnissa kukin valittu verkkopalvelualusta asennetaan työkaluineen ja otetaan käyttöön. Kullakin verkkopalvelualustalla tehdään pienehkö sovellus, jotta voidaan arvioida alustan käyttöä ja sen ominaisuuksia. Arvioinnissa kokeillaan taulukossa 1 mainittuja toimintoja.

Koesovelluksena käytettiin Davisin ja Rudolphin (2010) kuvaamaa RaceTrack-verkkopalvelua, jossa urheiluseura voi julkistaa tulevia juoksukilpailuja internetissä ja seuran jäsenet voivat ilmoittautua osallistujiksi. Sovelluksessa käytetään kaikkia arvioitavia ominaisuuksia jollakin tavalla. Sovellukseen liittyvät luokat *Race*, joka kuvaa kilpailuja, *Runner*, joka kuvaa kilpailijoita sekä luokka *Registration*, joka yhdistää kilpaili-

jan kilpailuun. Sovelluksessa on tarpeen mukaan käytetty erillistä luokkaa sovellukseen kirjautumista varten.

CRUD-näkymät (Create, Read, Update, Delete) voidaan luoda toiminnalla, josta käytetään nimitystä scaffolding. Käytettävä sovellusalusta tai työkalu on voinut luoda mallinäkymät, joita käyttäen scaffold luo valituista tai kaikista sovelluksen luokista CRUD-näkymät. JSON (JavaScript obect notation) on tekniikka, jossa objektien kentät ja kenttien tieto siirretään tekstimuodossa verkon yli. Esimerkkinä JSON-muodosta koesovellukseen syötetty kilpailu näyttää seuraavalta (päivämäärä *startDate* ASP.NET-muodossa):

```
{"raceID":2,"version":1,"name":"Urjalan yöjuoksu",
"startDate":"\/Date(1342731600000)\/","city":"Urjala",
"distance":19.2,"cost":25,"maxRunners":30}
```

TAULUKKO 1. Verkkopalvelualustojen arvioitavat toiminnot

Alkuun pääseminen	
Työkalut, asennus	Asennetaan alusta ja tarvittavat työkalut ja huomioidaan mahdollisia asennuksessa ilmeneviä seikkoja
Aloituksen helppous, ohjeet, esimerkit	Tutkitaan ohjeita ja esimerkkejä
Koesovellus - RaceTrack	
Tietokannan luonti ja tuonti	Sovellus luo tietokannan taulut objekteista (code first), sovellus luo objektit tietokannan tauluista (database first).
CRUD	Luodaan objektille CRUD-näkymät (luo, lue, päivitä, poista).
Suojaus	Luodaan 3 käyttäjää (tietokantaan) ja rajoitetaan sovelluksen näkymiä käyttäjän mukaan.
JSON-rajapinta	Luodaan objektille JSON-rajapinta Ajax- tai mobiilisovellusta varten.
Lokalisointi	Lisätään sovellukseen toinen kieli ja vaihto kielten välillä.
Omat näkymät	Lisätään sovellukseen uusi näkymä, joka voi pohjautua CRUD-näkymään. Muutetaan näkyvien tietojen määrää ja järjestystä, muutetaan syöttölomakkeella näkyvien kenttien määrää.

2.3 Alustojen valinta

Wikipedia listaa kaikkiaan 146 eri verkkosovellusalustaa (Wikipedia 2012b). Näistä alustoista suuri osa on Java- tai PHP-pohjaisia. Näistä lukuisista vaihtoehdoista seuraavat alustat tulivat valituiksi tämän työn ehdokkaiksi. Ehdokkaiksi valittiin yrityksen linjan mukaisesti Java-pohjaisia alustoja sekä muista ASP.NET MVC 3 ja Grails.

- ASP.NET MVC 3
- Spring (+ Roo)
- Apache Struts 2
- Apache Wicket
- JavaServer Faces
- Play! Framework
- Vaadin
- Google Web Toolkit
- Grails.

Ehdokkaiden ominaisuuksia on kuvattu liitteessä 1. Varsinainen vertailtavien alustojen valinta rajautui Java-pohjaisista alustoista Springiin Roo-työkalulla varustettuna. Spring ja Roo oli jo ennen vertailtavien alustojen valintaa valittu tuntikirjaussovelluksen toteutustavaksi. ASP.NET MVC 3 edustaa Microsoftin näkemystä avoimesta verkkopalvelualustasta ja työkaluista (Visual Studio). Grails on pienempi alusta, jota kuitenkin haetaan enemmän Googlen hakukoneella kuin Struts 2:lla, Wicketillä tai Play! Frameworkilla (Google trends). Google Web Toolkit on osana Rosenbergin opinnäytetyötä ja jätettiin tämän työn ulkopuolelle. Vaadin on mielenkiintoinen alusta, jota voi myöhemmin tutkia Springin lisäosana.

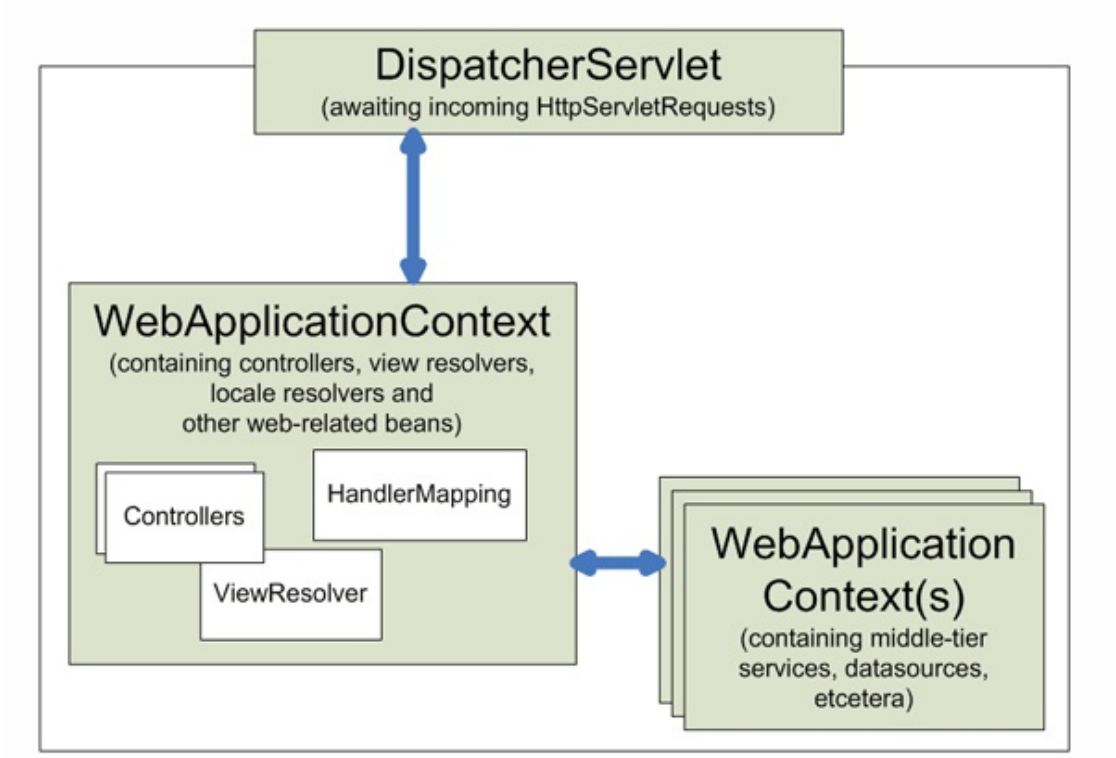
Kaikille valituille alustoille yhteistä on se, että ne mahdollistavat nopean alkuun pääsyn luomalla tietokantayhteyksiä, CRUD-näkymiä ja käyttäjän tunnistuksen tai helpottamalla niiden luomista. Alustat tai työkalut tekevät kehittäjän puolesta mekaanista perus- tai taustaohjelmointia (boilerplate code) erilaisten toteutusmallien perusteella. Kehittäjä pääsee näin nopeasti tekemään varsinaiseen sovellukseen kuuluvaa työtä, kuten sovelluksessa käytettäviä näkymiä sekä sovelluksen logiikkaa.

2.4 Spring

Spring on Java/J2EE-pohjainen (Java 2 Platform, Enterprise Edition) verkkopalvelualusta, joka ei käytä EJB-tekniikkaa (Enterprise JavaBeans). Spring sai alkunsa Rod Johnsonin ja Juergen Hoellerin työstä (Johnson & Hoeller 2004) kehittää tavallisia Java-luokkia (POJO, Plain old Java object) käyttävä verkkopalvelualusta ja siitä on tullut EJB:tä yksinkertaisempana hyvin suosittu. Spring ja Grails kuuluvat molemmat

SpringSource-yritykselle, joka on ollut VMwaren osana vuodesta 2009. Spring on edennyt versioon 3.

Springin tärkeimmät ominaisuudet ovat riippuvuuksien syöttö (dependency injection) ja aspektipohjainen ohjelmointi (aspect-oriented programming). Edellisessä objektit eivät itse määrää riippuvuuksiaan toisiin luokkiin, vaan sen tekee ajonaikaisesti erillinen komponentti, joka täyttää riippuvuudet tarpeen mukaan. (Walls 2011.) Jälkimmäisessä läpileikkaavat ominaisuudet (esimerkiksi poikkeusten käsittely) siirretään tai toteutetaan muun ohjelmiston rinnalle aspekteihin (Virta 2009). Springissä toteutettavia luokkia ei yleensä tarvitse periyttää Springin luokista tai merkitä annotaatiolla. Springin ja Spring-sovelluksen arkkitehtuuri näkyy kuviossa 1.



KUVIO 1. Spring-sovelluksen arkkitehtuuri. (Johnson ym. 2010)

Spring Roo on tekstipohjainen työkalu, jonka tarkoituksena on luoda ohjelman perusosat kehittäjän puolesta ja antaa kehittäjän keskittyä ohjelmoinnissa olennaisiin asioihin (Long & Mayzak 2011). Spring Roo luo muun muassa entiteettien (luokka jonka voi hakea ja tallentaa tietokantaan) tietokantayhteydet sekä JavaBean-mallin mukaiset get- ja set-funktiot. Näkymiä luotaessa Spring Roo luo valmiit kontrollerit ja näkymäsivut CRUD-näkymiin. Käytännössä Spring Roolla voi luoda verkkosovellukselle toimivan pääkäyttäjänäkymän alle tunnissa.

Spring Roo voi luoda käyttöliittymän joko Spring-ympäristön omalla Web MVC -näkymämallilla, Googlen GWT:llä tai Vaadin-ympäristöllä. Muita verkkopalvelinalustoja, joita voidaan käyttää Spring-ympäristön käyttöliittymänä, ovat JavaServer Faces, Struts, WebWork ja Tapestry (SpringSource 2007).

2.4.1 Työkalut, asennus

SpringSource Tool Suite asennetaan yhdellä asennustiedostolla. Työkalu sisältää Eclipse-pohjaisen kehitysympäristön, Springin, Spring Roon sekä kehitysaikaisen Tomcat-serverin. Työkalun lisäksi pitää tarvittaessa asentaa Java Development kit. 64-bittisessä käyttöjärjestelmässä sekä SpringSource Tool Suite että Java Development Kit on oltava samaa versiota (32- tai 64-bittinen) (Lippert 2012). Lopuksi lisätään ympäristön polkuun asennettujen Mavenin ja Spring Roon bin-hakemistot.

Kehitysympäristö mahdollistaa verkkosovelluksen toiminnan seuraamisen ja virheenkorjauksen. Koska kehitysympäristö pohjautuu Eclipse-työkaluun, vaatii se laitteistolta melko paljon resursseja toimiakseen sujuvasti. Käytännössä minimivaatimuksena voi pitää Core i5 -luokan suoritinta ja 8 Gt keskusmuistia. Kehitysympäristö toimii vaatimattomammallakin laitteistolla, mutta esimerkiksi T5600-suorittimella ja 4 Gt keskusmuistilla varustetulla kannettavalla tietokoneella verkkopalvelimen käynnistys, sovelluksen lataaminen ja käynnistäminen kestää puolesta minuutista kahteen minuuttiin muista käynnissä olevista sovelluksista ja vapaan muistin määrästä riippuen.

Windows 7 -käyttöjärjestelmässä SpringSource Tool Suite on parasta asentaa johonkin muuhun hakemistoon kuin Program Files tai Program Files (x86). Spring Roo (lähinnä käännöstyökalu Maven) tekee asennushakemistoihin muutoksia (lataa komponentteja verkosta) ja oletuksena sillä ei ole oikeutta tehdä niin. Jos vahinko on jo tapahtunut, voi kaikille SpringSource Toolsuite -asennushakemiston alihakemistoille antaa käyttäjälle kaikki oikeudet.

2.4.2 Aloituksen helppous, ohjeet, esimerkit

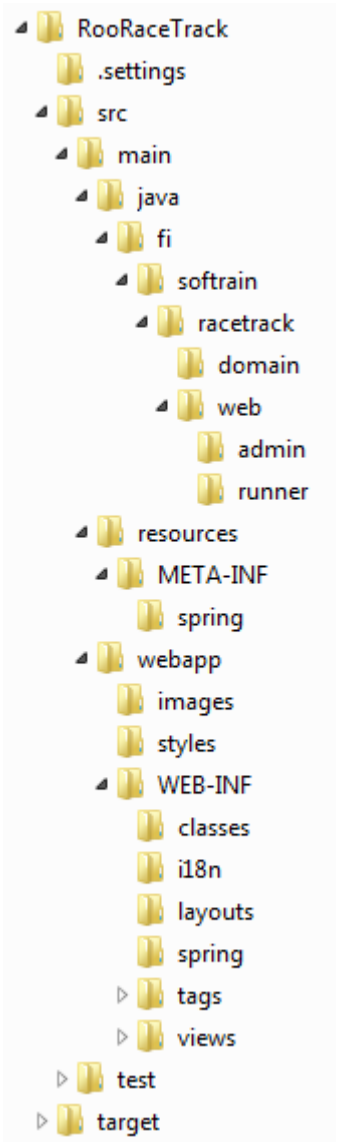
Spring Roo -työkalulla ohjelmoinnin aloittaminen on varsin helppoa. Spring Roon käytön oppii melko nopeasti ja sitä on helppo käyttää. Esimerkkejä on jonkin verran, tosin suuri osa niistä on melko yksinkertaisia. Hankalampiin vaiheisiin, kuten salasanan vaihtonäkymien tai omannäköisten sivujen luomiseen asti, etenevät vain harvat esimerkit tai kirjoitukset (esimerkiksi Mkyong 2010).

Näkymissä käytettävä merkintäkieli (XML tag) on hyvin tiivistä ja monien käytettyjen kenttien takia myös epäselvää. Näkymien rakenteen selvittämisen jälkeen eri kenttien järjestystä pystyy muuttamaan helposti, mutta muilla tavoilla niiden muuttaminen on silti hankalaa. Spring Roo palauttaa näkymien muutetut rivit takaisin entiselleen, ellei käyttäjä erikseen merkitse niitä tarkoituksella muutetuiksi.

Jos ohjelmaa muutetaan sen käynnissä ollessa, muutoksesta riippuen voi riittää näkymän lataaminen uudelleen (muutokset jspx-sivuilla tai tag-merkinnöissä) tai sovelluksen automaattinen lataaminen ja uudelleenkäynnistäminen palvelimella (muutokset kontrollereissa). Verkkopalvelimen manuaalinen uudelleenkäynnistäminen on tarpeen joissakin muutoksissa (muutokset sovelluksen asetustiedostoissa, käännöstiedostoissa, kenttien muutos entiteeteissä).

Spring-ympäristön keskusteluryhmistä voi löytää vastauksen omiin kysymyksiin, mutta valitettavasti monet kysymykset jäävät kokonaan ilman vastausta. Näyttää siltä, että StackOverflow on parempi lähde etsiä vastauksia ongelmiin, jos vain hakulauseen saa riittävän täsmälliseksi. Spring-ympäristöä ja Roota käsittelevää kirjallisuutta on saatavilla hyvin. Kirjasta riippuen osa toiminnoista voidaan esitellä hyvin pintapuolisesti ja kirjoihin kannattaa tutustua etukäteen mahdollisimman tarkasti.

Koesovelluksen hakemistorakenne näkyy kuviossa 2. Kuvioon on merkitty tärkeimmät hakemistot sekä joidenkin mainittujen tiedostojen sijainti.



KUVIO 2. Spring Roo -sovelluksen hakemistorakenne

2.4.3 Tietokannan luonti ja tuonti

Tietokantaa varten oli koneelle asennettu jo aiemmin MySQL 5.5. Sovelluksen käyttämä tietokantajärjestelmä määritetään Spring Roo -komennolla `persistence setup` listauksen 1 rivin 2 mukaisesti. Komennolla annetaan käytettävä tietokanta, tietokantayhteys salasanoineen sekä käytettävä rajapinta (tässä Hibernate). Tietokannan ja sen yhteystiedot voi lisätä myös käsin *database.properties*-tiedostoon. MySQL-tietokantaa käytettäessä tietokanta ja sen käyttäjä voidaan helposti lisätä komentoriviä käyttäen. Kaikki syötetyt komennot tallentuvat *log.roo*-tiedostoon myöhempää tarkastelua tai uudelleenkäyttöä varten.

```
001   project --topLevelPackage fi.softrain.racetrack --projectName
            RooRaceTrack --java 6
002   persistence setup --provider HIBERNATE --database MYSQL
            --databaseName "rooracetrack" --userName "roo"
            --password "roo"
```

LISTAUS 1. Projektin luominen ja tietokantayhteyden lisääminen projektiin

Tietokannan luomisen ja määrittämisen jälkeen on vuorossa tietokantaluokkien eli entiteettien luonti (code first). Entiteeteille lisätään tarvittava määrä kenttiä, joiden tyyppi ja rajoitteet määritellään tässä vaiheessa. Kenttiä voi muuttaa myöhemminkin hyvin vapaasti. Jos entiteetin rakenne muuttuu oleellisesti, sitä vastaava taulu tietokannassa korvataan uudella, kun sovellus käynnistyy seuraavan kerran. Komennolla `focus` listauksen 2 rivillä 8 asetetaan määrätty entiteetti aktiiviseksi ja seuraavat kentät lisätään siihen.

Riveillä 14–17 haetaan tietokannasta (database first) yksi taulu, josta luodaan entiteetti (Long & Mayzak 2011, 15). Ensimmäinen komento epäonnistuu, koska ympäristössä ei vielä ole MySQL JDBC -ajuria. Seuraavilla komennoilla(rivit 15 ja 16) haetaan ja asennetaan ajuri Roon komentorivillä antamien ohjeiden mukaisesti ja toistetaan entiteetin luominen. Esimerkissä luotiin tietokantaan taulu *race*, jonka Roo toi sovellukseen kenttien pituuksineen ja niine rajoituksineen, jotka näkyvät tietokannan taulussa. Taulun rakenne otettiin Grailsin luomasta samannimisestä taulusta (katso luku 2.6.3).

Lopuksi muutetaan tiedostosta *persistence.xml* muuttujan *value* arvo *validate* arvoon *update*, jotta taulut luodaan tietokantaan ja muutosten myötä päivitetään. Tämä muutos on ohjeistettu *persistence.xml*-tiedoston kommenteissa.

```
<property name="hibernate.hbm2ddl.auto" value="validate"/>
```

Verkkosovelluksen lopullisessa, julkisesti käytettävässä versiossa, on hyvä palauttaa muuttujan arvoksi *validate*. Jos sovelluksen tietorakenteissa parannusten myötä tapahtuu muutoksia, voidaan päivityksen ajaksi asettaa arvo *update* tai muuttaa tietokannan rakennetta käsin.

```
001   entity --class ~.domain.Runner
002   field string --fieldName firstName --notNull --sizeMax 50
003   field string --fieldName lastName --notNull --sizeMax 50
004   field date --type java.util.Calendar --fieldName dateOfBirth
            --notNull
005   enum type --class ~.domain.Gender
006   enum constant --name M
007   enum constant --name F
008   focus --class fi.softrain.racetrack.domain.Runner
009   field enum --type fi.softrain.racetrack.domain.Gender –fieldName
            gender --notNull
010   field string --fieldName address
011   field string --fieldName zipCode
012   field string --fieldName city
013   field string --fieldName email –regexp
            "[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}"
014   // [failed] database reverse engineer --schema rooracetrack
            --package ~.domain
015   addon info id --searchResultId 01
016   addon install id --searchResultId 01
017   database reverse engineer --schema rooracetrack --package
~.domain
018   entity --class Registration
019   field reference --type fi.softrain.racetrack.domain.Race
            --fieldName race --cardinality MANY_TO_ONE
020   field reference --type fi.softrain.racetrack.domain.Runner
            --fieldName runner --cardinality MANY_TO_ONE
```

LISTAUS 2. Tietokantaobjektien lisääminen Spring-projektiin

2.4.4 CRUD

Yksinkertainen CRUD-näkymä luodaan helpoiten kahdella komennolla: `web mvc setup` ja `web mvc all -- package ~.web`. Nämä komennot luovat kaikille entiteeteille kontrollerit samaan web-hakemistoon ja näkymät views-hakemiston alihakemistoihin. Jotta näkymien rajaaminen käyttäjän oikeuksien mukaan olisi myöhemmin helpompaa, käytettiin tässä CRUD-näkymien luontiin `web mvc scaffold` -komentoa, jolla voidaan määrätä kontrollerin hakemisto ja polku (Long & Mayzak 2011, 24).

```
001    web mvc setup
002    web mvc scaffold
              --class fi.softrain.racetrack.web.RunnerController
              --backingType fi.softrain.racetrack.domain.Runner
              --path admin/runner
003    web mvc scaffold –class fi.softrain.racetrack.web.RaceController
              --backingType fi.softrain.racetrack.domain.Race
              --path admin/race
004    web mvc scaffold –class
              fi.softrain.racetrack.web.RegistrationController
              --backingType fi.softrain.racetrack.domain.Registration
              --path admin/registration
```

LISTAUS 3. verkkokäyttöliittymän lisääminen Spring-projektiin

Näiden komentojen jälkeen peruskäyttöliittymä on muuten valmis, mutta Spring Roo 1.1.5 jättää luomaansa *load-scripts.tagx*-tiedostoon kaksi virhettä (SpringSource 2011). Tiedostosta tulee korjata seuraavat rivit poistamalla korostetut osuudet tai siirtämällä ne rivien loppuun.

```
<link rel="stylesheet" type="text/css" href="${tundra_url}"><!--
required for FF3 and Opera --></link>
<link rel="stylesheet" type="text/css" media="screen" href="$
{roo_css_url}"><!-- required for FF3 and Opera --></link>
```

Kuten jo luvussa 2.4.2 mainittiin, Spring Roo käyttää näkymiin varsin tiivistä merkintäkieltä. Listauksessa 4 esitetään osa koesovelluksen näkymätiedostosta, joka esittää kilpailut taulukossa riveillä. Taulukon kenttä data viittaa kontrollerin lähettämään tietopakettiin nimeltä races (kontrollerissa haetut kilpailut). Jokaisella rivillä on kenttä *z*, johon Spring Roo on laskenut muista kentistä tarkistussumman. Jos tarkistussumma ei täsmää, Spring Roo palauttaa rivin oman mallin mukaiseksi. Jos kentälle *z* antaa arvon *"user-managed"*, Spring Roo jättää kyseisen rivin rauhaan. Lisäksi kullekin riville tulee *<table:column>* -merkinnästä toiminnot kunkin kilpailun tarkastelemiseksi, muuttamiseksi ja poistamiseksi.

```
001   <table:table data="${races}" id="l_fi_softrain_racetrack_domain_Race"
            path="/admin/race" z="xgB/WQQWOzdc0M7yU1xaGQuXSAc=">
002         <table:column
                  id="c_fi_softrain_racetrack_domain_Race_city"
                  property="city" z="NJHDUwdmQEDOY5ga8PzR/2eZls4="/>
003         <table:column
                  id="c_fi_softrain_racetrack_domain_Race_cost"
                  property="cost" z="CEgnGm9kHU3LGjNsUJZCwU9gOcA="/>
004         <table:column
                  id="c_fi_softrain_racetrack_domain_Race_distance"
                  property="distance" z="NjVEdDlMQm28Dnj6F0iaVPOQyFw="/>
005         <table:column
                  id="c_fi_softrain_racetrack_domain_Race_maxRunners"
                  property="maxRunners" z="icfGe9S5Hrs6W6mrFZiOds2wWGc="/>
006         <table:column
                  id="c_fi_softrain_racetrack_domain_Race_name"
                  property="name" z="PMd9k9qfGzSbdfFBYYN0kM8KbI0="/>
007         <table:column date="true"
                  dateTimePattern="${race_startdate_date_format}"
                  id="c_fi_softrain_racetrack_domain_Race_startDate"
                  property="startDate" z="JPzEXgkUodSgYKpP3jOFaffFX1M="/>
008   </table:table>
```

LISTAUS 4. Spring Roon luomaa näkymää

2.4.5 Käyttäjien hallinta ja näkymien rajoittaminen

Springin tavallisin lisäosa käyttäjien hallintaan on Spring Security ja se asennetaan komennolla `security setup`. Komento luo asetustiedoston *applicationContext-security.xml*, jota muokkaamalla voidaan luoda kiinteitä käyttäjiä, asettaa käyttää- tai roolikohtaisia rajoituksia http-polkujen näkyvyyteen sekä määrittää tietokantapohjainen käyttäjien tunnistus (Long & Mayzak 2011, 44). Http-polkuihin asetetut rajoitukset vaativat sen, että eri oikeuksille sallitut näkymät on luotu omiin http-polkuihinsa, kuten luvussa 3.2 kerrotaan. Käyttäjien näkymiä voidaan rajoittaa myös kontrollereissa ja kontrollerien metodeissa @PreAuthorize("hasAuthority('ROLE_ADMIN')") -annotaatioilla (Alex & Taylor, 20). Tällä tavalla rajoitettua näkymää ei pysty saamaan luvatta esille, vaikka tietäisi suojatun toiminnan polun.

Käyttäjälle näkyviä osioita ja valittavissa olevia toimintoja voidaan rajoittaa jspx-sivuilla (Java Server Pages XML-muodossa) <security:authorize access="hasRole

('ROLE_ADMIN')"> … </security:authorize> -merkinnällä. Edellisen merkinnän sisäpuolella olevat toiminnot näkyvät vain käyttäjälle, jolla on ROLE_ADMIN rooli (Mak 2008, 498).

Käyttäjän tunnistus tietokannassa voidaan hoitaa ainakin kahdella tavalla, joko hakemalla suoraan tietokannasta tai toteuttamalla security.core.userdetails.UserDetails-rajapinta. Molemmilla tavoilla käyttäjän entiteetissä tulee olla kentät käyttäjätunnukselle, salasanalla ja käyttäjätunnuksen voimassa olemiselle. Ensimmäisellä tavalla käyttäjän tiedot haetaan suoraan tietokannasta kahdella hakulauseella, kuten tuntikirjaussovelluksen yhteydessä kappaleessa 3.2 tarkemmin selostetaan.

Toisella tavalla käyttäjäentiteetti toteuttaa UserDetails-rajapinnan määräämät metodit. Lisäksi luodaan uusi luokka, joka toteuttaa security.core.userdetails.UserDetailsService-rajapintaa. Tämä luokka hakee UserDetails-rajapinnan täyttävän käyttäjän, vaikkapa Spring Roon luoman hakufunktion UserInfo.findUserInfoesByUserNameEquals avulla ja palauttaa tämän käyttäjän ylemmälle taholle. Jos käyttäjän tunnistus käyttää salasanan tiivistettä, tulee käyttäjäluokan tehdä salasanan tiiviste joko kontrollerissa tai käyttäjäentiteetissä silloin, kun uusi käyttäjä luodaan, tai kun käyttäjän salasana vaihtuu. (Alex & Taylor, 29.)

2.4.6 JSON-rajapinta

Spring kontrollerit voivat vastaanottaa ja palauttaa JSON-muotoisia tietoja. Kontrollerin metodi palauttaa tiedon JSON-muodossa, kun metodin paluutyypiksi merkitään @ResponseBody. Kontrollerin metodi ottaa vastaan tietoa JSON-muodossa, kun tiedon lähteeksi merkitään @RequestBody ja pom.xml-tiedostoon lisätään riippuvuus Jackson-JSON-muuntimeen. (Donald 2010.) Rajapinnan käyttö kuvataan tarkemmin tuntikirjaussovelluksen yhteydessä luvussa 4.1. Listauksessa 5 esitetään yksinkertaiset esimerkit, jotka palauttavat kaikki kilpailut (rivi 3), tietyn kilpailun (rivi 8) tai syötetyn kilpailun (rivit 12 ja 13).

```
001   @RequestMapping(value="/race", method = RequestMethod.GET)
002   public @ResponseBody List<Race> getRaces() {
003         return Race.findAllRaces();
004   }
005
006   @RequestMapping(value="/race/{id}", method = RequestMethod.GET)
007   public @ResponseBody Race getRace(@PathVariable("id") Long id) {
008         return Race.findRace(id);
009   }
010
011   @RequestMapping(value="/race", method = RequestMethod.POST)
012   public @ResponseBody Map<String, ? extends Object>
            SaveRace(@RequestBody Race race) {
013         return Collections.singletonMap("race", race);
014   }
```

LISTAUS 5: Spring JSON palautus ja vastaanotto

2.4.7 Lokalisointi

Sovelluksen käännökset tehdään tiedostoihin *application.properties* ja *application_XX.properties,* sekä *messages.properties* ja *messages_XX.properties*, joissa *XX* on käännöksen kielen lyhenne, esimerkiksi *application_fi.properties*. Oletuskielisissä tiedostoissa ei ole kielen lyhennettä. Käytössä olevaa kieltä voi vaihtaa parametrillä *lang=XX*. Valittu kieli tallentuu evästeeseen siihen asti, kunnes kieli erikseen vaihdetaan. (VMware 2011.)

Spring Roo sisältää komennon `web mvc install language --code XX`, joka luo käännöstiedostojen pohjat annetulla kielellä, kopioi maan lipun hakemistoihin ja asettaa käännöksen valittavaksi sivulle (Gulati 2012). Komento toimii kuudelle eri kielelle, ei kuitenkaan suomen kielelle. Kieliä voi lisätä itse, mutta tätä ei kokeiltu. Kielen vaihto voidaan toteuttaa vaikkapa *header.jspx*-tiedostoon listauksessa 6 kuvatulla tavalla ja hakemalla, tai piirtämällä, sopivan kokoinen lippu oikeaan hakemistoon.

```
001    <span id="language">
002           <spring:message code="global_language"/>
003           <c:out value=": "/>
004           <util:language label="Suomeksi" locale="fi"/>
005           <util:language label="English" locale="en"/>
006    </span>
```

LISTAUS 6. Spring kielen valinta

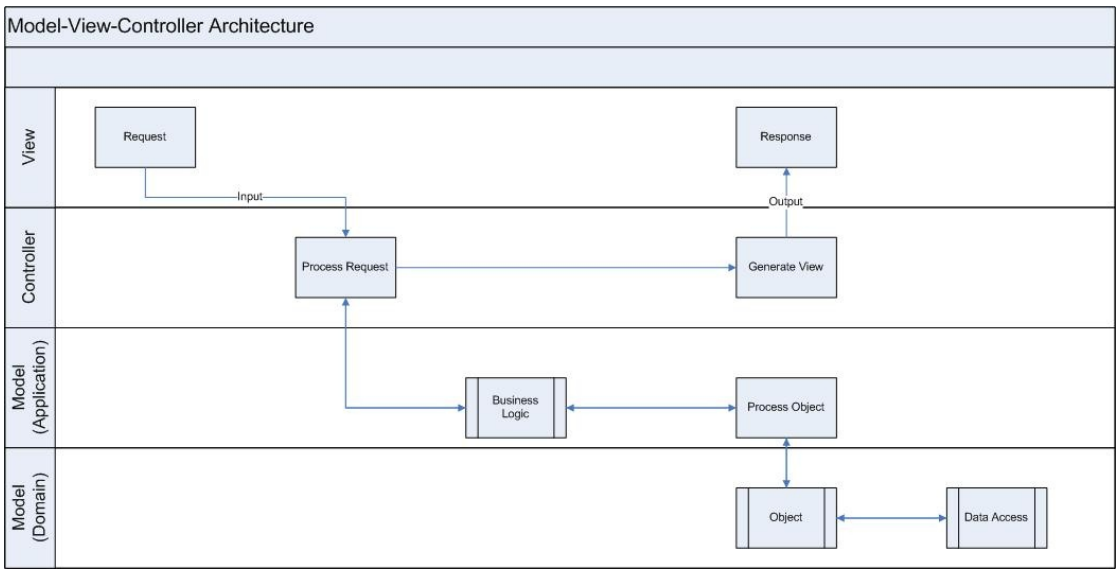
2.4.8 Omat näkymät

Omia näkymiä voi helpoiten luoda käyttämällä `web mvc scaffold` -komennolla luotua kontrolleria ja näkymätiedostoja pohjana. Kontrolleri irrotetaan Spring Roon hallinnasta, jotta Roo ei yrittäisi korjata tiedostoa, ja kontrolleria muokataan tarpeen mukaan. Näkymistä voi piilottaa käytön kannalta tarpeettomia kenttiä. Sovellukseen voi luoda myös erillisiä näkymäluokkia, mutta `web mvc scaffold` -komento luo tällaisille luokille vain tyhjän kontrollerin. Omista näkymistä kerrotaan tarkemmin tuntikirjaussovelluksen yhteydessä luvussa 3.4.

2.5 ASP.NET MVC

ASP.NET MVC on Microsoftin avoimen lähdekoodin MVC-mallin mukainen verkkopalvelualusta, joka on tehty ASP.NET-alustalle. ASP.NET MVC sai alkunsa vuonna 2007, kun Microsoftin Scott Guthrie luonnosteli ASP.NET MVC:n ytimen matkalla ALT.NET-konferenssiin. ASP.NET MVC 1.0 julkistettiin vuonna 2009, ja uusia versioita on tullut suunnilleen vuosittain. (Galloway, Haack, Wilson & Allen 2011.)

ASP.NET MVC 3:n tärkeimmät ominaisuudet ovat MVC-malli, url-osoitteiden reititys ja yksinkertaistaminen, standardien mukainen HTTP ja HTML sekä lisensointi. Kehittäjää ei myöskään ole sidottu yksittäisiin komponentteihin. Esimerkiksi entiteettien yhteys tietokantaan voidaan toteuttaa Microsoftin Entity Frameworkilla, NHibernatella tai jollain muulla ORM-työkalulla. Url-osoitteiden reitityksellä tarkoitetaan mallia, jossa haettava tieto yksilöidään http-parametrien (esimerkiksi */App_v2/User/Page.aspx?action=show%20prop&prop_id=82742*) sijaan url-polussa (esimerkiksi */to-rent/chicago/2303-silver-street*). Tekniikka on palvelun käyttäjälle edullinen ja hakukoneet voivat suosia yksinkertaisia osoitteita. (Freeman & Sanderson 2011.) Toki sama tekniikka on käytössä sekä Spring MVC:ssä että Grailsissa.



KUVIO 3. ASP.NET MVC -arkkitehtuuri. (Ayrenj 2008)

2.5.1 Työkalut, asennus

ASP.NET MVC:n asennus tapahtuu asentamalla Visual Studio. Opiskelu- ja harraste-käyttöön voi käyttää ilmaista Express-versiota, Visual Web Developer Express (Microsoft 2012c). Asennukseen kuuluu myös Microsoft SQL Server Express. Tässä työssä käytettiin Visual Studio 2010 Professional -versiota, joka ladattiin Microsoftin DreamSpark -sivustolta (Microsoft 2012b).

Kun halutaan käyttää MVC versiota 3, kuten tässä työssä, tarvitaan edellisten lisäksi ASP.NET MVC Tools Update, joka asentaa MVC-version 3 (Microsoft 2012a). Luotaessa uutta web-projektia MVC 3 tulee valittavaksi, kun ensin käytettäväksi .NET versioksi valitaan 4.0 tai uudempi. Visual Studio pitää sisällään kehitysaikaisen verkkopalvelimen, joka mahdollistaa verkkosovelluksen toiminnan seuraamisen ja virheenkorjauksen.

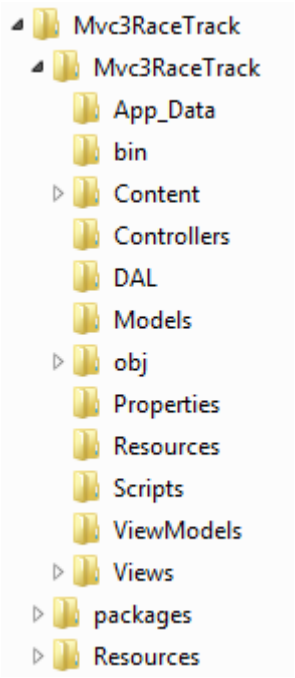
2.5.2 Aloituksen helppous, ohjeet, esimerkit

ASP.NET MVC 3 -ympäristöllä alkuun pääsy on helppoa. Yksinkertaisia esimerkkejä on useita, ja niitä seuraamalla ympäristön toiminta selviää melko helposti. Näkymissä käytetään hyvin väljää merkintäkieltä ja esimerkiksi taulukon rivit muodostavat silmukat on helppo havaita. Näkymiä on myös helppo muuttaa, mutta entiteettien muutoksia ei voi automaattisesti tuoda näkymätiedostoihin.

Kirjallisuus ja keskusteluryhmät antavat useimpiin kysymyksiin riittävän vastauksen, pienenä poikkeuksena ehkä sovelluksen kääntäminen eri kielille, josta lisää luvussa

2.5.7. Microsoftin luokkadokumentaatio sisältää myös esimerkkejä luokkien ja metodien käytöstä.

ASP.NET MVC 3 -sovelluksen hakemistorakenne esitetään kuviossa 4. Springistä poiketen hakemistorakenne on selkeästi jaettu entiteetteihin (Model), kontrollereihin (Controller) ja muihin tiedostoihin, sen sijaan että hakemistorakenne heijastaisi luokkien nimiavaruutta.



KUVIO 4. ASP.NET MVC -sovelluksen hakemistorakenne

2.5.3 Tietokannan luonti ja tuonti

Tässä koesovelluksessa käytettiin ORM-työkaluna Entity Framework -työkalua, joka yhdistää entiteetit tietokantaan. Kun tietokanta luodaan entiteeteistä (code first), luodaan ensin entiteettiluokat kenttineen, rajoitteineen ja keskinäisine yhteyksineen. Sopivaan hakemistoon (tässä DAL-hakemisto) luodaan System.Data.Entity.DbContext-luokasta periytyvä kontekstiluokka. Tähän kontekstiluokkaan lisätään kokoelma (System.Data.Entity.DbSet) kullekin tallennettavalle entiteettiluokalle. Lopuksi *web.config*-tiedostoon lisätään tietokannan yhteysrivi (connection string), jolle annetaan sama nimi kuin kontekstiluokalle, jolloin kontekstiluokka ja tietokantayhteys liittyvät toisiinsa. Jos sovelluksessa käytetään MicroSoftin SQL-tietokantapalvelinta, tietokantayhteys muodostuu automaattisesti ilman yhteysriviä.

Tällä tavoin luotu yhteys tietokannan ja kontekstiluokan välillä toimii siten, että tietoja haetaan ja tallennetaan kontekstiluokan kokoelmien kautta. Tietoja voidaan hakea monipuolisesti LINQ-kyselyillä, esimerkiksi 5 uusinta kilpailua lauseella

```
List<Race> lastRaces = db.Races.OrderByDescending(r => r.startDate).Take(5).ToList();
```

Tallennuksessa uusi objekti lisätään kokoelmaan ja kokoelman muutokset tallennetaan tietokantaan. Esimerkiksi uusi osallistuminen tallennetaan seuraavalla tavalla:

```
db.Registrations.Add(reg);
db.SaveChanges();
```

Sovelluksen testaamista varten voidaan tietokannan sisältö alustaa sovelluksen käynnistyessä, jos tietokanta muuttuu. Tämä tapahtuu System.Data.Entity.DropCreateDatabaseIfModelChanges-luokasta periytetyn alustajaluokan avulla. (Dykstra 2011, Hanselman 2011, 28.)

Tietokannasta voi Entity Frameworkia käyttäen myös tuodaan tauluja entiteeteiksi (data first). Mekanismi on hieman monimutkaisempi ja siitä voi lukea tarkemmin Julie Lermanin artikkelista (Lerman 2011). Kokeilussa tuotiin sovelluksen tietokantaan uusi taulu race, joka tuotiin sovellukseen. Tuonnin yhteydessä taulun id-kentästä ei tullut auto-increment-tyyppistä eikä sitä saanut sellaiseksi muokattua. Taulujen tuomista MySQL-tietokannasta ei kokeiltu.

Entiteetteihin voidaan lisätä erilaisia annotaatioita, jotka ohjaavat tietojen tarkistusta ja näkymistä CRUD-näytöillä. Annotaatioita ovat mm. [Required], joka määrittää pakollisen kentän, [HiddenInput(DisplayValue = true)], joka näyttää kentän arvon ilman että sitä pystyy muokkaamaan (DisplayValue = false piilottaa kentän) ja [StringLength(max*, MinimumLength=min*)], jolla määritetään kentän suurin ja tarvittaessa myös lyhin pituus.

Entiteettien keskinäisistä yhteyksistä on hyvä muistaa, että yksi-moneen-viittauksissa entiteetti sisältää kaksi kenttää viittausta varten. Esimerkkisovelluksessa viittaukset näyttävät listauksen 7 mukaisilta. Toinen kentistä on viittaus entiteetin tunnisteeseen (int raceID) ja toinen entiteettiin itseensä (virtual Race race). Vastaavasti toisessa

entiteetissä voi olla kokoelma siihen viittaavia olioita (virtual ICollection<...). Viittaukset entiteetteihin ovat virtuaalisia eli niiden sisältö haetaan vasta, kun sitä tarvitaan (lazy loading). (Galloway ym. 2011, 71.)

```
Registration.cs:
001    public int raceID { get; set; }
002    public virtual Race race { get; set; }
003    public int runnerID { get; set; }
004    public virtual Runner runner { get; set; }

Race.cs:
001    public virtual ICollection<Registration> registrations
              { get; set; }
```

LISTAUS 7. JSON-rajapinta Grails-kontrollerissa, tietojen vastaanotto

2.5.4 CRUD

Kun kontekstiluokka ja kaikki entiteetit kenttineen ja annotaatioineen on luotu, voidaan kontrollerit ja niitä vastaavat näkymät luoda automaattisesti Visual Studion Controllers-hakemiston komennolla `Add → Controller`. Kontrollerille annetaan nimi, entiteetti jota kontrolleri käyttää ja kontekstiluokka sekä valitaan, missä laajuudessa kontrolleri ja näkymät luodaan. (Freeman & Sanderson 2011, 35, Dykstra 2011.)

ASP.NET MVC 3 ei oletuksena luo pohjoismaisittain yhteensopivia syöttökenttiä desimaaliluvuille. MVC 3 ei hyväksy pilkulla erotettuja desimaalilukuja, mutta toisaalta kentän tarkistus ei hyväksy myöskään pisteellä erotettuja desimaalilukuja. Ongelma johtunee siitä, että JQuery-tarkistus selaimessa ja luvun tulkitseminen sovelluksessa eivät käytä samaa lokalisaatiota. Ongelma korjataan syöttämällä selaimen käyttämä lokalisointi molemmille osille. (Scala 2012.)

2.5.5 Käyttäjien hallinta ja näkymien rajoittaminen

ASP.NET sisältää käyttäjien hallinnan lähes valmiina. Kehittäjän tarvitsee lähinnä lisätä tietokantayhteys *web.config*-tiedostoon ja lisätä käyttäjän rajoitukset näkymiin ja kontrollereihin. MVC 3 käyttää sisäisiä käyttäjä- ja rooliluokkia (System.Web.Secu-

rity.Membership ja System.Web.Security.Roles), joissa käyttäjien ja roolien luominen ja yhdistäminen käyttäjiin tapahtuu. (Freeman & Sanderson 2011, luku 22.)

Kokeilussa käytettiin käyttäjän tunnistamiseen MySQL-tietokantaa. Esimerkin (Dykstra 2011) perusteella sovelluksessa käytettiin muiden tietojen tallennukseen sisäistä tiedostopohjaista MS SQL Server Compact Edition -tietokantaa. Käyttäjän hallintaa ei tällä tietokannalla kuitenkaan saatu toimimaan. MySQL-kytkentä luodaan muokkaamalla membership-, profile- ja roleManager-asetuksia listauksen 8 mukaiseksi (MySQL 2012).

Edellä mainitussa alustajaluokassa voidaan alustaa myös käyttäjien roolit (MSDN 2012c), jotta niitä ei tarvitse erikseen syöttää sovelluksen tai palvelimen hallintasivun kautta:

```
if (!Roles.RoleExists(RaceTrackContext.ROLE_USER))
      Roles.CreateRole(RaceTrackContext.ROLE_USER);
```

Uudelle käyttäjälle voi `AccountController`-kontrollerissa lisätä roolin seuraavalla tavalla (Labunskiy 2011):

```
Roles.AddUserToRole(model.UserName, RaceTrackContext.ROLE_USER);
```

```
001    <connectionStrings>
002          <add name="MySqlMembershipConnection"
connectionString="Data Source=localhost; userid=mvc3; password=mvc3;
database=mvcracetrack;" providerName="MySql.Data.MySqlClient" />
003    </connectionStrings>
...
004    <membership defaultProvider="MySqlMembershipProvider">
005          <providers>
006                <add name="MySqlMembershipProvider"
type="MySql.Web.Security.MySQLMembershipProvider, MySql.Web,
Version=6.4.4.0, Culture=neutral, PublicKeyToken=c5687fc88969c44d"
connectionStringName="MySqlMembershipConnection" ... />
007          </providers>
008    </membership>
009    <profile>
010          <providers>
011                <add name="MySqlProfileProvider"
type="MySql.Web.Security.MySQLProfileProvider, MySql.Web,
Version=6.4.4.0, Culture=neutral, PublicKeyToken=c5687fc88969c44d"
applicationName="/" connectionStringName="MySqlMembershipConnection"
autogenerateschema="true" />
012          </providers>
013    </profile>
014    <roleManager enabled="true" defaultProvider="MySqlRoleProvider">
015          <providers>
016                <add name="MySqlRoleProvider"
type="MySql.Web.Security.MySQLRoleProvider, MySql.Web, Version=6.4.4.0,
Culture=neutral, PublicKeyToken=c5687fc88969c44d"  applicationName="/"
connectionStringName="MySqlMembershipConnection"
autogenerateschema="true" />
017          </providers>
018    </roleManager>
```

LISTAUS 8. ASP.NET MySQL käyttäjän hallinta

2.5.6 JSON-rajapinta

ASP.NET MVC 3:ssa JSON-rajapinnan toteuttaminen on helppoa. Tietojen palauttaminen JSON-muotoisena tapahtuu palauttamalla JsonResult-tyyppinen arvo metodilla Json(*objekti*). Tietojen tallentaminen tapahtuu automaattisen muunnoksen kautta. (Strumpflohner 2011.) Listauksessa 9 esitetään yksinkertainen tietojen vastaanotto *Race*-luokan olioon sekä saman olion palauttaminen JSON-muotoisena.

```
001    [HttpPost]
002    public JsonResult Create(Race race)
003    {
004           return Json(race);
005    }
```

LISTAUS 9. JSON-rajapinta, tietojen vastaanotto ja palautus

JSON-rajapinnan rajoitteena on sovelluksen turvallisuus. Oletuksena kontrolleri ei voi palauttaa JSON-muotoista tietoa get-metodilla. Json-muunnokselle voidaan antaa lisäparametri return Json(race, *JsonRequestBehavior.AllowGet*) jonka jälkeen get-metodia voi käyttää. Sovelluksen on kuitenkin hyvä olla suojattu, tai palautettavien tietojen niin rajattuja, tai yleisluontoisia, että niiden paljastumisesta ei ole haittaa. JSON-muunnos palauttaa olion kaikki kentät. (Freeman & Sanderson 2011, 417.)

Toinen JSON-rajapinnan rajoite ovat virtuaaliseksi määritellyt kentät (lazy loading). Jos jokin kenttä on määritelty virtuaaliseksi, on tuloksena kehäviittaus ja ajon aikainen virhe (*System.InvalidOperationException: A circular reference was detected while serializing an object of type ...*). Entiteettien rakennetta voi JSON-rajapinnan takia muuttaa siten, että niissä entiteeteissä, joita JSON-rajapinnan kautta käsitellään, ei ole virtuaaliseksi määriteltyjä kenttiä. Toinen tapa korjata kehäviittaus on kieltää lazy loading, mutta tätä tapaa ei saatu kokeiltaessa toimimaan. (Slauma 2012.) Kolmas, ja ilmeisesti paras tapa ohittaa virtuaalisuuden rajoitteet, on käyttää nimetöntä apu- tai näkymäluokkaa, johon alustetaan JSON-rajapinnassa välitettävät kentät (Rahien 2012). Apuluokan ansiosta alkuperäiseen entiteettiin ja sen näkymiin ei tarvitse tehdä muutoksia, ja näkymäluokkaa täytettäessa voidaan rajata rajapinnan kautta siirrettävät kentät. Listauksessa 10 palautetaan kaikki kilpailut JSON-muotoisena. Avuksi luodaan uusi lista rivillä 3. Listaan liitetään kustakin kilpailusta luotu näkymäluokka riveillä 4 - 14, ja luotu lista palautetaan JSON-muotoisena rivillä 15.

```
001   public JsonResult Index()
002   {
003          List<object> lista = new List<object>();
004          foreach (Race race in db.Races.ToList())
005                 lista.Add(new
006                 {
007                        race.raceID,
008                        race.name,
009                        race.startDate,
010                        race.city,
011                        race.distance,
012                        race.cost,
013                        race.maxRunners
014                 });
015          return Json(lista, "application/json",
                   JsonRequestBehavior.AllowGet);
016   }
```

LISTAUS 10. JSON-rajapinta, tietojen valinta ja virtuaalisuuden välttäminen

2.5.7 Lokalisointi

ASP.NET MVC 3 sovelluksen käännökset tehdään resurssitiedostoihin. Käännösten käyttämistä ei vähäisten, ja vaillinaisten, ohjeiden perusteella saatu otettua käyttöön, joten ASP.NET-sovellusten kääntämistä eri kielille pitää tutkia myöhemmin. Tutkituista kirjoista yksikään ei selvittänyt käännösten toteuttamista.

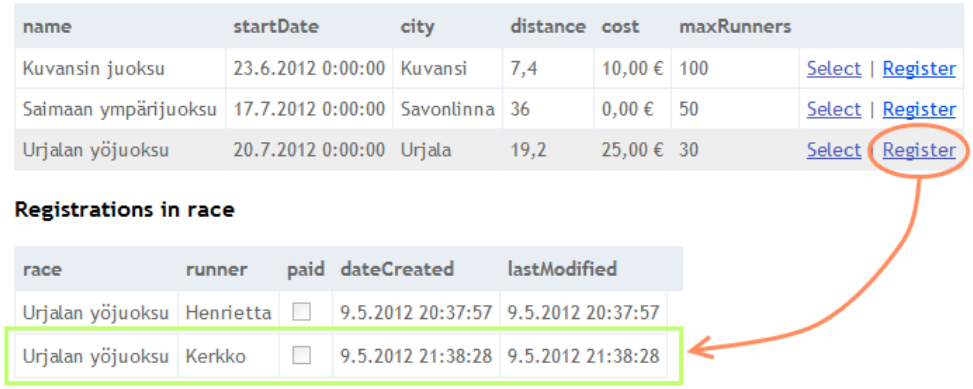
2.5.8 Omat näkymät

ASP.NET MVC 3 sovelluksen omia näkymiä voi luoda vastaavalla tavalla kuin Spring Roossakin: muokataan lähinnä vastaavasta kontrollerista ja näkymistä oikeanlainen käyttöliittymä. Kokeeksi luotiin Dykstran esimerkkisovellusta (Dykstra 2011) mukaillen kilpailunäkymä, jossa näkee valitsemansa kilpailun osallistujat, ja voi itse ilmoittautua kilpailun osallistujaksi. Osallistuminen ja osallistujien tarkastelu on rajoitettu vain kirjautuneille käyttäjille. Kilpailuja voivat lisätä, poistaa ja muokata vain pääkäyttäjät.

Näkymässä käytetään hyväksi näkymämallia (view model, Galloway ym. 2011, 45, Freeman & Sanderson 2011, 179). Näkymämalliin haetaan ja tallennetaan eri tyyppisistä entiteeteistä näkymään liittyvät tiedot. Kilpailunäkymässä entiteetit ovat kilpailu (*Race*) ja osallistumiset (*Registration*). Muille kuin pääkäyttäjälle osallistumisen

sijasta näkymään kuuluisi oikeastaan kilpailijan tiedot, koska se, onko osallistuminen maksettu, ei kuulu muille kilpailijoille.

Näkymässä on yhdistetty esimerkin mukaisella tavalla normaali kilpailujen listanäkymä, ja osallistumisien listanäkymä. Kontrollerissa on myös vastaavalla tavalla toteutettu kilpailuun liittyvien osallistumisien haku, ja liittäminen näkymämalliin. Kilpailuun osallistuminen on toteutettu poimimalla kilpailun ja käyttäjän tunnisteet. Jos näillä tiedoilla ei löydy aiemmin lisättyä osallistumismerkintää, luodaan niiden perusteella uusi osallistuminen, joka tallennetaan tietokantaan. Kuvassa 1 esitetään näkymä sekä lisätty osallistuminen.



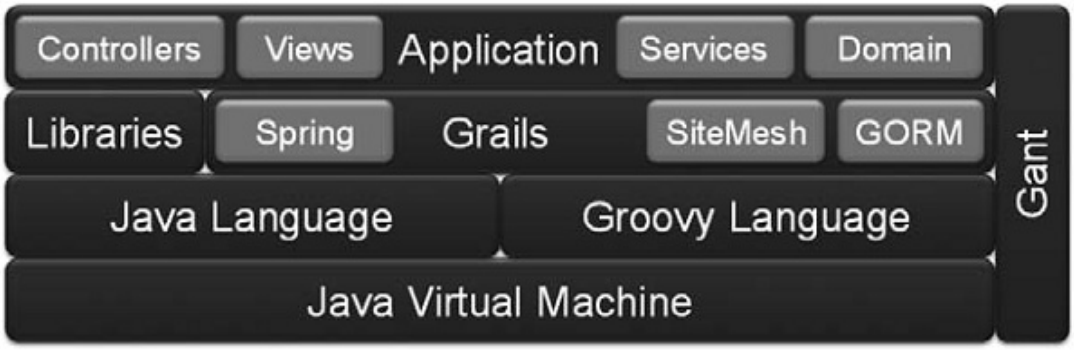
KUVA 1. Kilpailuun ilmoittautuminen

2.6 Grails

Grails on avoimen lähdekoodin verkkopalvelualusta, jossa käytetään Groovy-ohjelmointikieltä. Groovy puolestaan pohjautuu Javaan, ja tästä johtuen Grails-sovellukset voidaan ladata tavalliselle Java-palvelimelle (esimerkiksi Tomcat tai Jetty). Grailsin kehittäminen alkoi vuonna 2005 nimellä *Groovy on Rails* mutta nimi muutettiin Ruby on Rails-kehittäjien pyynnöstä Grailsiksi. Grailsilla ei alkuperäisestä nimestään huolimatta ole yhteyttä Ruby on Railsiin.

Grailsin tärkeimmät ominaisuudet ovat itsenäinen kehitysympäristö, mahdollisuus käyttää olemassa olevia Java-kirjastoja ja automaattiset toiminnot. Automatiikasta esimerkkeinä mainittakoon entiteettien hakutoiminnat, tietokantatoiminnallisuus ja automaattiset näkymät. Esimerkiksi Spring Roo luo vastaavat toiminnat lähdekoodiin, ja osan vain erikseen komennettuna. *Convention over configuration*, jossa halutut asetukset tai toiminnallisuus liitetään konfiguraatiotiedostojen sijasta lähdekoodiin,

kuuluu Grailsin lisäksi myös muihin vertailtaviin verkkopalvelualustoihin. Esimerkiksi kontrollerin totutuksessa url-polku ja sovellus/kontrolleri/toiminto vastaavat toisiaan ilman erillistä konfigurointia (esimerkiksi kontrollerikohtainen servlet-mapping). (Smith & Ledbrook 2009.)



KUVIO 5. Grails arkkitehtuuri. (Judd, Nusairat & Shingler 2008)

2.6.1 Työkalut, asennus

Tässä työssä Grailsia käytettiin NetBeans 7.1 ohjelmointityökalun kanssa. NetBeansiin asennetaan Groovy and Grails lisäosa. Groovyn (Groovy 2012) asennustiedosto ladataan ja puretaan haluttuun hakemistoon, samoin tehdään Grailsille (Grails 2012). Molempien binäärihakemistot lisätään käyttöjärjestelmän polkuun ja kotihakemistot asetetaan ympäristömuuttujiin *GROOVY_HOME* ja *GRAILS_HOME*.

Tämän jälkeen uusi sovellus luodaan valitsemalla projektivalinnoista Groovy ja projektin tyypiksi Grails sovellus. Groovyn ja Grailsin asennushakemistot asetetaan NetBeansille *Configure Grails* -ikkunassa, kun Grails-projekti luodaan ensimmäisen kerran. (NetBeans 2012.)

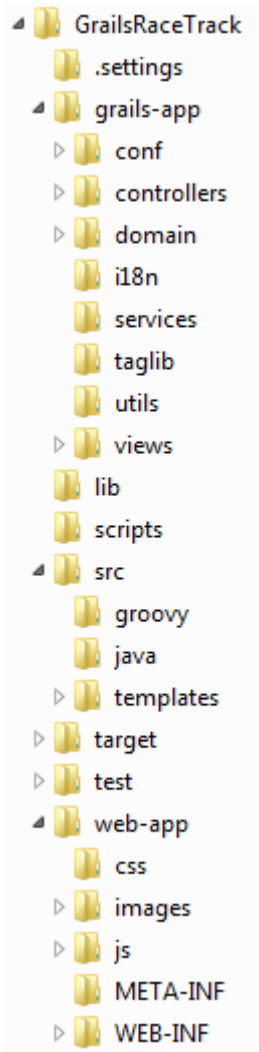
Aluksi kokeiltiin uutta Grails versiota 2.0.2, mutta kohdattujen ongelmien takia päädyttiin vanhempaan versioon 1.3.8. Uudempi versio ei esimerkiksi toiminut oikein automaattisen dateCreated-kentän (joka nimensä mukaisesti asettuu, kun entiteetti luodaan) kanssa, ja palautti virheen syöttölomakkeelta puuttuvista parametreista, vaikka parametrien lähetys selaimelta varmistettiin selaimen verkkotyökalulla. Lisäksi ympäristö ilmeisesti suoritti vanhoja käännöksiä mm. kontrollereista sovelluksen muutoksista ja kääntämisistä huolimatta.

2.6.2 Aloituksen helppous, ohjeet, esimerkit

Grails on työkaluna samantapainen kuin Spring Roo, ja myös sitä voi käyttää suoraan komentoriviltä. Grails on hyvin integroitu NetBeansiin ja monia toimintoja voi suorittaa käyttöliittymässä hiiren avulla. Spring Roossa entiteettien kentät voitiin luoda joko käsin tai Roota käyttäen, Grailsissa vain käsin. Kentät kirjoitetaan käytännössä kahteen kertaan, koska ensin esitellään kenttä ja kentän tyyppi, ja *constraints*-osiossa kentän rajoitteet sekä järjestys CRUD-näytöissä. Jos kenttä viittaa toiseen entiteettiin, voi kentälle tulla vielä kolmaskin merkintä.

Grailsin luomat näkymätiedostoissa käytetään gsp-merkintöjä (Grails server pages), ja ne ovat ASP.NET MVC 3:n tapaan selkeitä ja helposti ymmärrettäviä. Kontrollereissa ja sovelluksen logiikassa kielenä käytettävä Groovy poikkeaa Javasta ja C#:sta, ja vaatii jonkin verran opettelua. Mahdollisia olemassa olevia Java-luokkia ja toteutuksia voi käyttää Grails-sovelluksen osana, ja myös uutta toimintalogiikkaa voi Grails-sovellukseen lisätä Java-kielellä (Smith & Ledbrook 2009, 8).

Kirjallisuutta ja esimerkkejä on mukavasti, esimerkiksi Grailsin tutorials-sivu listaa kymmeniä johdatuksia ja artikkeleita. Esimerkeistä mainittakoon Davisin ja Rudolphin verkosta vapaasti ladattavissa oleva kirja *Getting Started with Grails* (Davis & Rudolph 2010), jonka esimerkkisovellusta *RaceTrack* on käytetty testisovelluksena tässä työssä arvioitavilla alustoilla.



KUVIO 6. Grails-sovelluksen hakemistorakenne

2.6.3 Tietokannan luonti ja tuonti

Grails luo tietokannan taulut ja kentät entiteeteistä automaattisesti, kun sovellus käynnistetään ensimmäistä kertaa (code first). GORM (Grails object relational mapping) toimii taustalla automaattisesti ja tarjoaa rajapinnat entiteettien hakemiseen ja tallentamiseen. Listauksessa 11 on lähes koko *Race*-luokan (kilpailu) toteutus. Entiteetin kenttien järjestys näytöllä (oletuksena aakkosjärjestys), sekä kenttien rajoitukset ja omat tarkastimet (mm. pakollinen kenttä rivillä 9 ja päiväys tätä hetkeä myöhemmin rivillä 10), määritellään entiteetin *constraints*-lohkossa. Jos jokin kenttä halutaan jättää pois tietokannasta, lisätään se *transients*-lohkoon. Tällaisia kenttiä käytetään ehkä käyttöliittymässä apukenttinä tai useamman kentän koosteena. (Davis & Rudolph 2010.)

Yksi-yhteen-sidoksissa riittää, että luokkaan lisätään kenttä, jonka tyyppi on toinen luokka. Yksi-moneen-sidoksissa kenttä lisätään muodossa static hasMany = [*kentännimi*:*ViitattavaLuokka*] (rivi 7). Jos sidos on kaksisuuntainen, toinen osapuoli määrittää viittauksen muodossa static belongsTo = [*kentännimi*:*ViittaavaLuokka*]. (Davis & Rudolph 2010.)

```
001    String name
002    Date startDate
003    String city
004    BigDecimal distance
005    BigDecimal cost
006    Integer maxRunners = 100000
007    static hasMany = [registrations:Registration]
008    static constraints = {
009           name(blank:false, maxSize:50)
010           startDate(validator: {return (it > new Date())})
011           city()
012           distance(min:0.0)
013           cost(min:0.0, max:100.0)
014           maxRunners(min:0, max:100000)
015    }
```

LISTAUS 11. Race-luokan toteutus

Grails ei itse osaa hakea tietokannan tauluja ja kenttiä entiteeteiksi (database first). Tarkoitusta varten on työkalu nimeltä The GRails Application Generator. Työkalulla pystyy ilmeisesti myös luomaan entiteetit UML mallista (model first). Mallin voi luoda esimerkiksi ArgoUML-ohjelmalla ja tallentaa XMI-muodossa. (Ekkelenkamp 2009.) Työkalua ei kokeiltu tämän työn yhteydessä.

2.6.4 CRUD

Yksinkertainen käyttöliittymä saadaan käyttöön luomalla kullekin entiteetille kontrollerit. Kontrollerin toteutukseen riittää yksi rivi, *def scaffold = true*. Grails luo sovelluksen suorituksen aikana kaikki tarvittavat näkymät ja kontrollerin toiminnot. (Davis & Rudolph 2010, 15–16; Judd, Nusairat & Shingler 2008, 91.) Vaihtoehtoisesti voidaan entiteetille luoda pelkkä näkymä (generate views, GSP-sivut), ja käyttää automaattista kontrolleria tai toteuttaa kontrolleri itse, tai luoda koko käyttöliittymä (generate all, GSP-sivut ja kontrollerien toteutus) (Davis & Rudolph 2010, 66).

Grails voi luoda automaattisesti yhtä tai kahta hakutermiä käyttäviä hakufunktioita ajon aikana. Hakufunktioiden pitää kuitenkin kohdistua yksittäiseen entiteettiin. (Smith & Ledbrook 2009, 106–109.) Esimerkiksi tulevien kilpailujen hakeminen voidaan suorittaa hakufunktiolla Race.*findAllByStartDateGreaterThanEquals*(new Date()), joka pitkän nimensä mukaisesti hakee kaikki kilpailut, jotka alkavat tänään tai tulevaisuudessa.

Hakufunktioita voi käyttää myös hakemaan esimerkin mukaisia tietoja. Hakufunktiolle annetaan esimerkkinä haettavan tyypin luokka, johon on täytetty haettavat kriteerit. Haku esimerkin mukaan voi hakea vain täsmälleen samanlaisia tietoja (Rocher, Ledbrook, Palmer, Brown, Daley & Beckwith 2012, luku 5.4.1; Smith & Ledbrook 2009, 110). Esimerkiksi kaikki Kaisa-nimiset kilpailijat voi hakea seuraavilla lausekkeilla:

```
def finder = new Runner(firstName: 'Kaisa')      // esimerkki
def runnerInstance = Runner.find(finder)         // hakulause
```

2.6.5 Käyttäjien hallinta ja näkymien rajoittaminen

Grailsiin on useita lisäosia käyttäjän hallintaan. Tässä sovelluksessa käytetään Spring-security -lisäosaa. Lisäosa asennetaan komennolla `grails install-plugin spring-security-core`, minkä jälkeen luodaan entiteetit ja sisäinen toteutus käyttäjille ja rooleille komennolla `s2--quickstart` *`pakkaus käyttäjäluokannimi rooliluokannimi`*. (Ledbrook 2010.) Jos sovellus ilmoittaa luokan *Person* puuttuvan (Person on Spring-security -lisäosan oletus käyttäjäluokalle), siirretään tiedostossa *Config.groovy* olevat grails.plugins.springsecurity-asetukset tiedoston alkuun (Neels 2011). Spring-security lisäosa käyttää salasanoihin oletuksena SHA-256 -tiivistefunktiota. Jos halutaan käyttää jotain muuta tiivistefunktiota, se voidaan määrätä *Config.groovy* tiedostossa. (Beckwith & Talbott 2012, 58.)

Käyttäjien toimintaa kontrollerilla tai kontrollerin metodilla voidaan rajata käyttämällä annotaatiota (Beckwith & Talbott 2012, 23)

```
@Secured(['ROOLI'])
```

Tällä annotaatiolla voidaan esimerkiksi estää kontrollerin metodien create, save, edit, update ja delete käyttö muilta kuin pääkäyttäjältä. Polkukohtainen rajoitus määritetään tiedostossa *config.groovy*. (Beckwith & Talbott 2012, 25.)

Käyttäjälle näkyviä toimintoja voi rajata gsp-sivuilla merkinnöillä <sec:ifLoggedIn> ja <sec:ifNotLoggedIn> sekä <sec:ifAllGranted roles="*ROOLI*"> (Beckwith & Talbott 2012, 29–30). Ensimmäisillä merkinnöillä rajataan näkyviä toimintoja sen perusteella, onko käyttäjä kirjautunut sovellukseen (käyttäjän oikeuksilla ei ole merkitystä). Esimerkiksi, jos käyttäjä on kirjautunut sovellukseen, näytetään seuraavan listauksen mukaisesti hänen käyttäjätunnuksensa sekä linkki sovelluksesta poistumiseen. Jos käyttäjä ei ole kirjautunut sovellukseen, näytetään hänelle linkki sovellukseen kirjautumiseen. Jälkimmäisellä merkinnällä käyttöliittymässä esillä olevia toimintoja rajataan kirjautuneen käyttäjän oikeuksien mukaisesti. Tällä tavalla voidaan vaikkapa käyttöliittymässä näkyviä muokkauspainikkeita piilottaa tavalliselta käyttäjältä, mutta näyttää ne pääkäyttäjälle.

```
001   <sec:ifLoggedIn>
002         <sec:username /> (<g:link controller="logout">
                  Sign out</g:link>)
003   </sec:ifLoggedIn>
004   <sec:ifNotLoggedIn>
005         <g:link controller="login" action="auth">Login</g:link>
006   </sec:ifNotLoggedIn>
```

LISTAUS 12. Esitettävien toimintojen rajaaminen kirjautumisen perusteella

2.6.6 JSON-rajapinta

Grailsin kontrollereissa JSON-rajapinnan käyttö on yksinkertaista. Kontrolleri palauttaa tietoja suoraan JSON-muodossa käyttämällä muotoa *as JSON*. Esimerkiksi render Runner.list() *as JSON* palauttaa kaikki kilpailijat JSON-muodossa. Jos palautettavat kentät halutaan valita itse, käytetään yksityiskohtaisempaa muotoa (näkymäluokka), joka on esitetty listauksessa 13. (Rocher ym. 2012.)

```
001   def list = {
002         def results = Race.list()
003         render(contentType:"text/json") {
004               races = array {
005                     for(r in results) {
006                           race name:r.name
007                     }
008               }
009         }
010   }
```

LISTAUS 13. JSON-rajapinta Grails-kontrollerissa, tietojen palautus

Grails voi myös vastaanottaa tietoja JSON-muodossa. Vastaanotto on samantyyppinen kuin XML-muotoiselle tiedolle (Smith & Ledbrook 2009, 316), mutta muunnos objektiksi on vielä hieman helpompi.

```
001   def jsonObject = request.JSON
002   def instance = new Runner(jsonObject)
```

LISTAUS 14. JSON-rajapinta Grails-kontrollerissa, tietojen vastaanotto

2.6.7 Lokalisointi

Sovelluksen käännökset tehdään *messages_XX.properties*-tiedostoihin, joissa *XX* on kunkin kielen lyhenne, esimerkiksi *messages_fi.properties*. Grailsin luomilla CRUD-gsp-sivuilla peruskäännöksiä käytetään automaattisesti. Kehittäjän lisäämiä käännöksiä käytetään muodossa <g:message code="*app.frontpage.welcome*" /> jota vastaava käännös messages_fi.properties-tiedostossa on esimerkiksi *app.frontpage.welcome*=Tervetuloa sovellukseen. (Rocher ym. 2012, luku 10.)

Sovelluksessa käytettävä kieli valitaan url-parametrilla *lang=XX*. Valittu kieli tallentuu evästeeseen ja määrää sovelluksessa käytettävän kielen siihen asti, kunnes se taas vaihdetaan toiseksi (esimerkiksi /GrailsRaceTrack/runner/show/2?*lang=fi*). Sovellukselle voidaan asettaa oletuskieli, jota käytetään siihen asti, kunnes käyttäjä erikseen valitsee jonkin kielen käytettäväksi. Listauksessa 15 asetetaan sovelluksen oletuskieleksi suomi. Grailsin perusvirheilmoitukset on käännetty valmiiksi 15 kielelle, ei kuitenkaan suomeksi.

```
001   beans = {
002         localeResolver(org.springframework.web.servlet.i18n.
                  SessionLocaleResolver) {
003               defaultLocale = new Locale("fi","FI")
004               java.util.Locale.setDefault(defaultLocale)
005         }
006   }
```

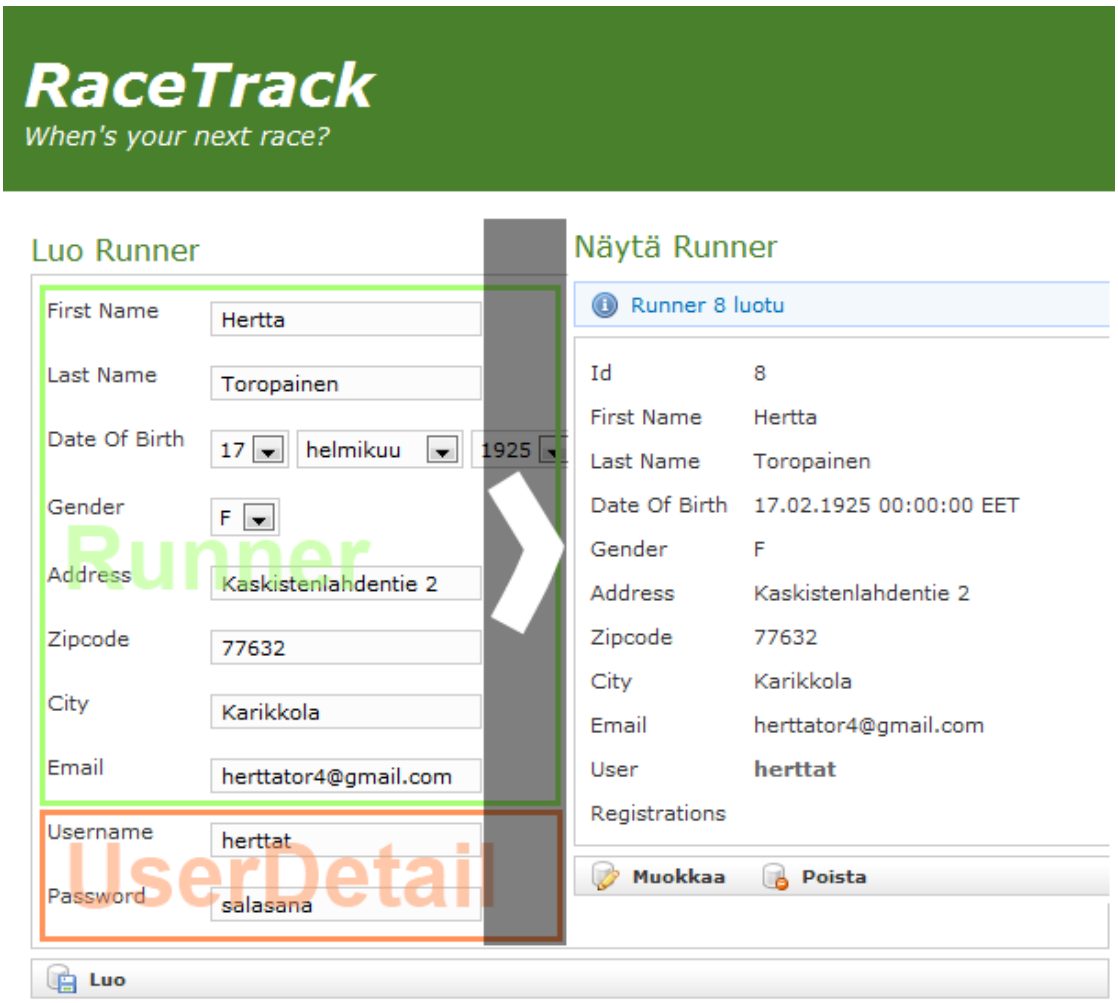
LISTAUS 15. Grails lokalisointi, oletuskielen asetus

2.6.8 Omat näkymät

Myös Grailsissa omia näkymiä voi tuottaa luomalla ensin kontrolleri ja sitä vastaavat näkymät ja muokkaamalla niitä. Toinen, monesti ehkä helpompi tapa, on luoda kontrolleri ja tarvittavat näkymät (gsp-tiedostot) itse. Omia <g: > -merkintöjä on helppo tehdä luomalla merkintää varten ensin kirjastoluokka ja sitten kirjastoluokkaan oma metodi kutakin merkintää varten. (Davis & Rudolph 2010.)

Uuden käyttäjän luominen on oma näkymä (kuva 2), joka toteutettiin. Näkymään on yhdistetty kaksi entiteettiä, *UserDetail* (käyttäjän kirjautumistiedot, ei vastaa Springin UserDetails-rajapintaa) ja *Runner* (käyttäjän henkilötiedot), jotka syötetään samalla kertaa. Näkymässä yhdistettiin molempien entiteettien syöttökentät. Kontrollerissa kummankin entiteetin tiedot luetaan kentistä, luodaan entiteetit ja tallennetaan ne toisiinsa viitaten. Kuvaan on vasemmalle puolelle (syöttäminen) merkitty entiteetit. Kuvan oikealla puolella näkyvät tallennetun juoksijan (Runner) tiedot, yhtenä niistä käyttäjän kirjautumistunnus *User*.

Ennen näkymien luomista voi Grailsiin ladata ja asentaa tiedostopohjat, joihin luotavat entiteetit, kontrollerit ja eri gsp-sivut pohjautuvat. Tiedostopohjia muokkaamalla voi sovelluksen ulkoasua ja kontrollerien perustoimintoja muuttaa halutunlaisiksi. Muokkausten jälkeen kukin luotu kontrolleri tai näkymä on muokatun pohjan mukainen, samoin kuin Grailsin automaattisesti luomat näkymät. (Davis & Rudolph 2010.)



KUVA 2. Grails, yhdistetty käyttäjän lisäys

3 TUNTIKIRJAUSSOVELLUS

3.1 Tuntikirjaussovellus

Tuntikirjaussovellusta ryhdyttiin suunnittelemaan ja kokeilemaan ilman erityistä verkkopalvelualustaa. Kontrollerit olisi toteutettu servletteinä, näkymät html- ja jsp-tiedostoina ja tietokantayhteys MySQL-tietokantaan suoraan servleteistä. Tässä vaiheessa verkkopalvelualustojen tutkimus ei vielä ollut alkanut, mutta ajatuksena oli verrata raa'an toteutuksen ja verkkopalvelualustojen käyttöä toisiinsa.

Samaan aikaan yrityksessä oli alkamassa myös toinen hanke, johon haettiin sopivaa toteutustapaa. Yhtenä ehdokkaana oli GWT, mutta muitakin vaihtoehtoja haluttiin tarkastella. Pikaisten internethakujen perusteella vaihtoehtoina olisivat voineet olla Spring, Vaadin tai GWT:n yhdistäminen Springiin. Tästä syntyi myös päätös toteuttaa tuntikirjaussovellus alusta lähtien käsin kirjoitettujen servlettien sijasta Spring-alustalle Roolla toteutettuna.

Koska tietokanta ja toiminnat oli jo suunniteltu, saatiin yksinkertainen CRUD-sovellus alkuun varsin nopeasti. Samalla huomattiin myös Spring Roon rajoitukset käyttöliittymän ulkonäössä ja toiminnoissa. CSS-tiedostoilla ulkonäköä pystyy toki muuttamaan jonkin verran. Tästä syystä ja toisesta yrityksessä meneillään olleen hankkeen vuoksi, Rosenberg alkoi tutkia ja kokeilla, millä tavalla Spring, Roo ja GWT oikeasti saadaan yhdistettyä toimivaksi kokonaisuudeksi. Tästä yhdistämisestä, tietokantasuunnittelusta ja käyttöliittymän suunnittelusta voi lukea Rosenberigin opinnäytetyöstä. (Rosenberg 2012.)

3.1.1 Näkymä, kontrolleri

Spring Roo luo näkymän ja ohjaimen samanaikaisesti. Spring Roo luo CRUD-näkymiä (**cr**eate, read, **u**pdate, **d**elete), joista voi jättää osan toiminnoista pois jo luontivaiheessa. Käyttöliittymän rakentaminen aloitetaan Spring Roon komennolla `web mvc setup`. Seuraavaksi voitaisiin antaa komento `web mvc all --package` *`sijainti`*, jossa sijainti on tavallisesti *~.web*.

Tässä sovelluksessa hakemistot on kuitenkin jaettu käyttäjätyyppien mukaan, jolloin on helpompaa hallita, minkä tyyppiset käyttäjät voivat käyttää mitäkin kontrollereita. Käyttäjätyypistä riippuu myös se, millä tavalla käyttäjät voivat nähdä ja muokata eri

entiteettien tietoja. Tällöin käytetään Spring Roo -komentoa `mvc scaffold --backingType malli --class kontrolleri --path http-polku` (esim. `web mvc scaffold --backingType ~.domain.Worker --class ~.web.admin.users --path admin/users`).

3.2 Suojaus ja käyttäjän tunnistus

Käyttäjien autentikoinnissa käytetään Spring Roo komentoa `security setup` (Long & Mayzak 2011, 44). Tämän jälkeen lisätään tietokantaan `entity workerrole`, jossa on kentät `name` ja `description`. Käyttäjälle lisätään kenttä `workerrole` (`field set --fieldName workerrole --type fi.softrain.tuntimaku.domain.Workerrole --cardinality MANY_TO_MANY`) johon voidaan määrittää useita rooleja kullekin käyttäjälle. Roolien nimet määräytyvät *applicationContext-security.xml*-tiedostossa, jossa eri roolien käytettävissä olevat url-polut (esimerkiksi <intercept-url pattern="/user/**" access="hasRole('USER')"/>) rajoitetaan. Samalla tavalla *menu.jspx*-tiedostossa eri valintojen näkyvyyttä rajoitetaan eri käyttäjätyypeille (esimerkiksi <security:authorize access="hasRole('USER')">).

Käyttäjän haku tehdään hakulausekkeella "`SELECT worker.username AS username, worker.password as password, worker.active as enabled FROM worker where worker.username=?`" ja käyttäjän roolien haku monimutkaisella lausekkeella "`SELECT worker.username as username, A.name as authority FROM worker left join worker_workerrole UA on worker.id=UA.worker left join workerrole A on UA.workerrole = A.id WHERE worker.username=?`" (Robert 2010).

Käyttäjät tallennetaan rooleinensa tietokantaan, josta heidät sittemmin tunnistetaan. Yhdellä käyttäjällä voi olla useampi rooli, koska esimies todennäköisesti kirjaa myös omat työtuntinsa järjestelmään. Koska käyttäjät tunnistetaan käyttäjätunnuksen perusteella, on käyttäjätunnus määritelty mallissa, ja tietokannassa, uniikiksi.

Kun käyttäjä kirjautuu sovellukseen, lähetetään salasana syöttölomakkeelta selväkielisenä palvelimelle. Lähetettävän salasanan suojausta yritettiin automatisoida salasanan satunnaistuksella eli suolauksella (salt-source) (Wheeler 2008) mutta tästä aiheutui kaksi ongelmaa. Ensimmäinen ja ratkaisematta jäänyt ongelma oli se, että applicationcontext-security.xml-tiedoston käyttämä security-nimiavaruus ei salli JavaBean-luokille suolauksen vaatiman ominaisuuden lisäystä (decoration). Toinen

ongelma oli se, että suolaus ei suojaisi salasanan lähetystä, koska salasanan suolaus tapahtuu vasta palvelimella.

Suojausta on nyt parannettu käyttämällä selaimen ja palvelimen välillä suojattua SSL-yhteyttä (Walls 2011, 234). Testattaessa SSL-yhteys toimi Tomcat-palvelimella moitteettomasti, mutta GlassFish-palvelin ei osannut vaihtaa tavallisesta yhteydestä suojattuun SSL-yhteyteen automaattisesti. Kyseessä oli luultavimmin kehityspalvelimen porttinumeroihin liittyvä ongelma, joka ratkennee lisäämällä asetuksiin porttien määrittelyn (Mularien 2010, 121).

Jatkossa on tarkoituksenmukaista muokata kirjautumissivua siten, että salasanasta ja suolasta (esimerkiksi käyttäjätunnus) lasketaan tiiviste JavaScript-kielisellä ohjelmalla selaimessa ennen lähettämistä.

TAULUKKO 2. Polkujen suojaus

Järjestys	Käyttö	pattern	access	requires-channel
1	Mobiilirajapinta	/mobileapi/**	permitAll	any
2	Kirjautuminen	/login	permitAll	https
3	Tuntien kirjaus	/user/**	USER	-
4	Projektien hallinta, kirjausten hyväksyminen	/supervisor/**	SUPERVISOR	-
5	Tilitoimiston näytöt	/accountant/**	ACCOUNTANT	-
6	Ylläpito	/admin/**	ADMIN	-
7	Etusivu	/**	permitAll	-

Palvelimen eri näkymät on rajattu käyttäjän roolin mukaan taulukon 2 mukaisesti. Tuntikirjaussovelluksessa on käytetty myös käyttäjän muistamista (remember-me). Kirjautumissivulla käyttäjä voi merkitä haluavansa palvelimen muistavan käyttäjän, minkä jälkeen palvelin tunnistaa käyttäjän kirjautuneena siihen asti, kunnes käyttäjä kirjaa itsensä pois palvelusta, tai edellisestä käytöstä on kulunut palvelimella asetettua pitempi aika. Käyttäjän muistaminen tallennetaan evästeeseen, a se otetaan käyttöön lisäämällä suojausasetuksiin rivi <remember-me user-service-ref = "*userService*"/>, sekä lisäämällä sama tunniste tietokannasta käyttäjän tiedot hakevaan osioon <jdbc-user-service id = "*userService*" data-source-ref="dataSource" (Raible 2011).

3.3 Käyttöönotto

Sovelluksella on yksi kiinteä käyttäjä *localadmin*, jolla on rajoitettu rooli *LOCALADMIN*, ja jolle annetaan yrityskohtainen salasana samalla, kun sovellus asennetaan. Tällä käyttäjällä on oikeus luoda tietokantaan tuntikirjaussovelluksen perustiedot (käyttäjäroolit, alustavat projektit, tehtävät ja vaiheet) sekä tietokantaan tallennettava pääkäyttäjä, joka suorittaa sovelluksen käyttöönoton loppuun. Pääkäyttäjä voi valita tekeekö hän itselleen kaksi erillistä käyttäjätunnusta (varsinainen pääkäyttäjä ja tavallinen käyttäjä) vai antaako hän itselleen useamman roolin (*ROLE_ADMIN* ja *ROLE_USER*). Pääkäyttäjä voisi syöttää projektit ja muut asetukset myös käsin, mutta käyttäjien oikeuksien hallinta riippuu oikein syötetyistä käyttäjärooleista.

Organisaation tulee myös suunnitella millä tavalla tehtäviä ja vaiheita käytetään. Tehtävät voivat olla esimerkiksi laskutuksen tai palkanlaskennan mukaisia kirjauskohteita tai suoritettavia työtehtäviä. Vaiheet voivat vastaavasti olla varsinaisia työvaiheita tai muita tarkentavia selitteitä. Palkanlaskennan raportointi luultavasti rajoittaa tehtävien ja vaiheiden käytön edelliseen malliin siten, että kukin tehtävä sisältää vain yhteen palkkaluokkaan kuuluvia tehtäviä.

3.4 Mukautetut näkymät

Spring Roo ei osaa luoda muita kuin CRUD-näkymiä. CRUD-näkymät sopivat oikein hyvin sovelluksen ylläpitoon, mutta varsinaiselle käyttäjälle ne ovat hieman karuja, ja esittävät usein asiat väärällä tavalla. Spring Roossa kehittäjä pystyy määräämään, mitä operaatioita CRUD-näkymä sisältää, muiden luonti voidaan kieltää. Kuhunkin CRUD-näkymään liittyy yksi kontrolleriluokka ja yksi hakemisto, jossa on jspx-sivuja eri toimintoja varten. Nämä toiminnat Spring Roo lisää myös menu.jspx-tiedostoon, joka näkyy sovelluksen vasemmassa reunassa ja josta toimintoja voi valita. Spring Roo voi myös helposti palauttaa käyttäjän tekemät muutokset jspx-sivuilla varoittamatta.

Ehkä helpoin tapa tehdä omia näkymiä on luoda sellainen CRUD-näkymä, joka on lähinnä tarkoitusta, poistaa näkymä Spring Roon hallinnasta ja sen jälkeen käsin editoida kontrolleria ja jspx-sivuja halutunlaiseksi. Spring Roo poistetaan yksittäisestä kontrollerista kopioimalla kontrollerin AspectJava-tiedostosta kaikki funktiot kontrollerin Java-tiedostoon joko käsin tai käyttämällä STS-komentoa Refactor → Push in ja poistamalla kontrollerin alusta rivi @RooWebScaffold. Viimeistään seuraavan kään-

nöksen yhteydessä Spring Roo (jos se on käynnissä) huomaa asian ja poistaa AspectJava-tiedostot ja jättää kontrolleriin liittyvät jspx-sivut rauhaan.

Esimerkki: käyttäjän tuntikirjaus

Käyttäjän tuntikirjauksesta on jspx-sivulta poistettu käyttäjän valinta ja sen tilalla näytetään käyttäjän nimi. Jotta tietojen tarkastus toimisi, on sivulle lisätty piilotettu syöttökenttä, jossa käyttäjän tunniste siirtyy kontrollerille tallennuksen yhteydessä. Kirjausta tallennettaessa tai muutettaessa tarkistetaan tehtävän alkuaika, loppuaika ja kesto. Jos annetuista ajoista ei pysty laskemaan tehtävän kestoa tai jos kirjaus menee toisen päälle, palautetaan edellinen näkymä virheilmoituksen kera. Jos kirjaus hyväksytään, kopioidaan projekti, tehtävä, vaihe ja kuvaus sekä edellisen loppuaika uuden kirjauksen alkuajan pohjaksi.

Esimerkki: käyttäjän tuntikirjausten lista jokaisella sivulla

GWT:n mallin mukaisesti kaikilla sivuilla näkyy sivutettu lista kaikista käyttäjän kirjauksista aikajärjestyksessä, ja valitun toiminnan mukaisesti uuden kirjauksen syöttö, vanhan kirjauksen muutos tai valitun kirjauksen yksityiskohdat. Listaa voi selata sivu kerrallaan eteenpäin ja taaksepäin, ja listalta voi valita kirjauksen muutettavaksi tai tarkasteltavaksi. Muutoksia on tehty kontrolleriin, joka hakee tiedot listaan, ja näkymien jspx- sekä tagx-tiedostoihin.

Esimerkki: esimies hakee ja hyväksyy tuntikirjauksia

Esimies voi hakea tuntikirjauksia käyttäjän ja alkuajan perusteella. Tätä hakua on muokattu, jotta se palauttaa kirjaukset sivutettuna aikajärjestyksessä. Hyväksyntää yritettiin ensin tehdä listana, josta voisi valita useamman kirjauksen, ja tallentaa ne kaikki yhdellä kertaa. Kun tämä ei onnistunut, ja edellä kuvattu käyttäjän tuntikirjausten lista jokaisella sivulla näytti hyvältä, sitä sovellettiin myös tuntien hyväksyntään. Koska hakuparametreja (käyttäjä ja aikajakso) ei pystynyt järkevästi välittämään näkymien ja kontrollerin välillä, niistä tehtiin tietokantaan oma taulu. Joka kerta, kun esimies hakee käyttäjän tunnit, mahdolliset vanhat hakutermit tuhotaan tietokannasta, ja uudet tallennetaan tilalle. Kullakin sivun latauksella hakuparametrit luetaan tietokannasta, ja niiden perusteella haetaan listan tiedot.

Esimerkki: pääkäyttäjä vaihtaa toisen käyttäjän salasanan

Pääkäyttäjä ei käyttäjien tietoa tarkastellessaan näe näiden salasanoja (oikeammin salasanojen tiivisteitä). Jopa käyttäjän tietoja muutettaessa salasanakenttien sisältö ei ole näkyvillä, vaan salasanamuodossa (***). Lisäksi salasanakentissä ei ole todellista sisältöä, vaan salasanan ja varmistuksen kentissä on tekstit ”##########” ja

”**********”. Kun käyttäjän muuttuneet tiedot tallennetaan, kontrolleri tarkistaa salasanakenttien tekstit. Jos molempiin kenttiin on syötetty uusi teksti, ja kenttien tekstit ovat yhtenevät, tallennetaan käyttäjän uudeksi salasanaksi pääkäyttäjän antaman salasanan tiiviste. Jos salasanakenttiä ei ole muutettu, ei käyttäjän salasanaa vaihdeta.

3.5 Suorituskyky

Tuntikirjaussovelluksen mitoituksen perusteeksi asetettiin laskennallinen 1800 tapahtumaa päivän aikana. Tapahtumien määrä koostuu sadan käyttäjän kirjauksista ja muusta käytöstä. Lisäksi sovelluksen vasteajaksi määriteltiin tavallisesti 5 sekuntia tai vähemmän, mutta korkeintaan 20 sekuntia, yhteyttä kohti. (Rosenberg & Toivanen 2011.) Voidaan ajatella, että sovelluksen suorituskyky joutuu suurimmalle rasitukselle kuukauden lopussa, kun harvemmin kirjauksia syöttäneet käyttäjät tallentavat tehtävänsä sovellukseen.

Tuntikirjaussovelluksen toimintaa mitattiin hyvin lyhyesti internetselaimen työkaluilla. Tuntikirjaussovellus oli toiminnassa T5600-suorittimella ja 4 Gt keskusmuistilla varustetulla kannettavalla tietokoneella (kehitysympäristö) ja mittaus tapahtui sisäverkossa toisella tietokoneelta. Mittaukset ovat karkeita mutta jossain määrin suuntaa antavia. Varsinaiset suorituskykymittaukset tulee kuitenkin suorittaa todellisessa palvelinympäristössä keinotekoisella kuormituksella palvelimen suorituskykymittareita hyväksi käyttäen.

Pisimmät vasteajat saatiin silloin, kun sovelluksen osia (näytä tunnit tai uusi kirjaus) käytettiin ensimmäistä kertaa. Vasteajat olivat tällöin näyttämiselle 4 sekuntia ja uudelle kirjaukselle noin seitsemän sekuntia. Seuraavilla kerroilla samat toiminnot kestivät alle sekunnin. Myös tuntiraportin saaminen kesti alle sekunnin. Tästä voidaan päätellä, että testattu kone ja ympäristö pystyy vastaamaan tunnissa noin 4000 yhteyspyyntöön. Jos yhteen tuntikirjaukseen vaaditaan kaksi yhteyspyyntöä (uusi tuntikirjaus ja tietojen lähetys) vastaa se noin 2000 tuntikirjausta tunnin aikana. Jos käyttäjä tekee yhdestä kuukaudesta 130 kirjausta (keskimäärin kuusi kirjausta / päivä), ja käyttää näiden kirjausten syöttämiseen tunnin mittaisen yhtäjaksoisen ajan, pystynee sovellus palvelemaan muutaman kymmenen käyttäjän huippukuormitusta saman tasoisella palvelimella ilman huomattavaa hidastumista.

4 MOBIILIKÄYTTÖ

Tuntikirjaussovellusta voi käyttää mobiililaitteiden selaimilla, ainakin jos sovellukseen tehtäisiin mobiililaitteiden selainten ja näyttöjen mukainen teema. Ainakin nykyinen teema on liian monimutkainen testatun Android-laitteen selaimelle, ja osa näkymästä jää piiloon. Jos tuntikirjaussovellus olisi toteutettu Vaadin-verkkopalvelualustalla, olisi mobiililaitteille voitu luoda oma sivustonsa (maksullisen) TouchKit-lisäosan avulla, ja mobiililaitteen selain olisi voinut tallentaa jonkin verran tietoja myös silloin, kun laite ei ole verkkoyhteydessä palvelimeen. (Tahvonen & Grönroos 2012.)

Myös Spring verkkopalvelualustassa on mobiiliominaisuuksia. Spring Mobile mahdollistaa käytettävän laitteen tunnistamisen selaimen perusteella ja käyttäjän ohjaamisen tuntikirjaussovelluksen mobiililaitteita varten suunniteltuun osioon. Spring for Android puolestaan tarjoaa ohjelmointikirjastot REST-asiakkaan (Representational state transfer) ja Spring-autentikoinnin toteuttamiseen Android-alustalla. (SpringSource 2012.) Valitettavasti kirjastoja ei ole saatavilla muille mobiilialustoille.

Parhaaksi vaihtoehdoksi jää siis erillisten mobiilisovellusten suunnittelu ja toteutus erikseen kullekin mobiilialustalle. Erillinen sovellus mahdollistaa mobiililaitteen tietovarastojen käyttämisen kirjausten tallentamiseen. Etuna tästä on kirjausten tallentaminen ilman verkkoyhteyttä. Samalla voidaan mobiilisovelluksen käyttöliittymä räätälöidä kyseisen mobiilialustan mukaiseksi.

Erillinen mobiilisovellus vaatii palvelimelle rajapinnan jossa tarvittava tieto siirtyy palvelimen ja mobiilisovelluksen välillä turvallisesti ja tehokkaasti.

4.1 Tiedonsiirto palvelimen ja mobiilisovellusten välillä

Tiedonsiirrossa tavallisimmat tekniikat ovat XML (Extensible Markup Language) ja JSON (JavaScript Object Notation, Crockford 2006). Koska sekä Spring MVC 3 (Jackson JSON, Jackson 2012a), Android (GSON, Google 2012a tai Jackson JSON) että Windows Phone 7 (DataContractJsonSerializer, MSDN 2012b) tukevat JSON-rajapintaa, valittiin se tiedonsiirtoon. Palvelimen toteutusta testattiin tietokoneella luomalla Java-sovellus (NetBeans) ja Windows Form -sovellus (C#, Visual Studio 2010). Kummallakin sovelluksella haettiin oikeanlainen tapa siirtää tietoa sovellusten välillä, sekä oikea tapa muuntaa objektit JSON-muotoiseksi ja päinvastoin.

Apuna käytettiin Online JSON Viewer -työkalua (Gabor 2012), jolla voi helposti tutkia JSON-dataa objektimuodossa, sekä Epoch Converter -työkalua (de Goul 2012), jolla selvitettiin millä tavalla päiväys ja aika siirtyy JSON-rajapinnassa.

KUVA 3. Online JSON Viewer ja tuntikirjaus

4.1.1 JSON Spring kontrollerissa

JSON-rajapinnan lisääminen Spring MVC 3:een on todella helppoa. Projektitiedostoon lisätään Jackson JSON-kirjasto esimerkin (Donald 2010) mukaisesti.

```
001    <dependency>
002          <groupId>org.codehaus.jackson</groupId>
003          <artifactId>jackson-mapper-asl</artifactId>
004          <version>1.6.4</version>
005    </dependency>
```

LISTAUS 16. Jackson-kirjaston lisääminen Spring-projektiin

Palvelimella toimivan kontrollerin palauttama arvo määritellään *@ResponseBody tyyppi* -annotaatiolla JSON-muotoiseksi. Lukupyynnöille paluuarvon tyyppi on joko yksinkertainen objekti (esimerkiksi tietty työntekijä public *@ResponseBody Worker* getWorker(..)) tai lista (esimerkiksi kaikki projektit public *@ResponseBody List<Project>* listProjects(...)). Muokkauspyynnöille paluuarvon tyyppi on *Map<String, ?*

extends Object> jossa voidaan palauttaa joko lisätty tai muutettu objekti tai virheilmoitukset. (Donald 2010.)

Spring lähettää objektit täydellisinä, joten esimerkiksi palvelimen palauttama tuntikirjaus-objekti pitää sisällään työntekijän kaikkine tietoineen (mukaan lukien salasanan tiiviste ja työntekijän projektit) ja kirjauksen hyväksyjän kaikkine tietoineen. Kontrollerilla kaikki tarpeettomat tiedot (salasanat ja roolit) pitää erikseen tyhjentää lähetettävistä tiedoista. Mobiilirajapinnassa voisi myös siirtää vain mobiilisovelluksen ja tuntikirjauksen tarvitseman tiedon samaan tapaan kuin Grails- ja ASP.NET MVC 3 -esimerkeissä käyttämällä JSON-annotaatioita.

Kontrollerin vastaanottama objekti muunnetaan JSON-muodosta automaattisesti, kun parametri on määritelty @*RequestBody*-annotaatiolla (esimerkiksi tuntikirjaus public @ResponseBody Map<String, ? extends Object> saveHour(@*RequestBody Workhours hour*, ...)) ja mobiilisovellus määrittää lähettäessään tiedot tyypiksi application/json;charset=UTF-8. Tällä tavalla saatua objektia ei voi automaattisesti tarkastaa @Validate-annotaatiolla, vaan objekti pitää tarkastaa erikseen. Tarkastamisen suorittava komponentti lisätään kontrolleriin ja alustetaan konstruktorissa:

```
001    private Validator validator;
002    @Autowired
003    public MobileInterface(Validator validator) {
004           this.validator = validator;
005    }
```

LISTAUS 17. Tarkastimen lisäys kontrolleriluokkaan

Tarkastus suoritetaan vastaanotetulle objektille ja virheet (poikkeamat JPA-annotaatiosta) voidaan lukea palautetusta objektilistasta:

```
Set<ConstraintViolation<Workhours>> failures =
            validator.validate(hours);
```

Mahdolliset virheet muunnetaan ConstraintViolation-objekteista selväkieliseksi listaksi ja palautetaan mobiilisovellukselle HTTP-virheen 400 (bad request) myötä. Palautettaessa virhelista muuntuu JSON-muotoon mobiilisovelluksen luettavaksi muodossa

{"*virheellinen kenttä 1*":"*virheen kuvaus 1*","*virheellinen kenttä 2*":"*virheen kuvaus 2*"}.

Onnistuneen kirjauksen poiston paluuarvo on muodossa {"id":*arvo*} jossa *arvo* on poistetun kirjauksen tunniste. Onnistuneen uuden tai muutetun kirjauksen tallennuksen paluuarvo on muodossa {"hours":{*kirjaus*}} jossa *kirjaus* on tallennettu kirjausobjekti. (Donald 2010.)

4.1.2 JSON Java-sovelluksessa

Koska java-sovelluksessa käytettiin samaa Jackson JSON kirjastoa (Jackson 2012b) kuin verkkopalvelussa, tietojen siirto näiden sovellusten välillä toimi helpoiten. Siirrettävät luokat (Worker, Project, Task, Phase ja Workhours) voitiin kopioida suoraan tuntikirjaussovelluksesta. Jäsenmuuttujista poistettiin tietokanta-annotaatiot ja luotiin get- ja set-funktiot (jotka Roo oli luonut erillisiin AspectJava-tiedostoihin).

Lähetettävät objektit voidaan muuntaa JSON-muotoon Jackson-kirjaston funktiolla org.codehaus.jackson.map.ObjectMapper.writeValueAsString(*objekti*) mutta palvelimen päässä skandinaaviset merkit tuottivat ongelmia. Korjauksena tähän kokeiltiin tekstin muuttamista html-koodattuun muotoon (Mohan 2011), mutta tällä tavalla skandinaaviset merkit tallentuivat myös verkkopalvelussa koodattuna (esim. T*ämä* → T*ä*m*ä*). Oikea tapa muuttaa objektit JSON-muotoon on muuttaa ne suoraan UTF-8 muotoiseksi funktiolla ObjectMapper.writeValueAsBytes(*objekti*) ja liittää se http-pyyntöön tavumuodossa ilman muita välivaiheita tai muutoksia.

Vastaanotettu JSON-data muunnetaan Java-sovelluksessa objektiksi Jackson-kirjaston funktiolla ObjectMapper.readValue(palvelimenvastaus, new TypeReference <*tyyppi*>() { }). Tässä komennossa tyyppi on sama kuin palvelimen palauttama tyyppi, kuten yksittäinen objekti (esimerkiksi Project) tai objektilista (List<Project>). (Jackson 2012b.)

4.1.3 JSON Windows Form -sovelluksessa

Windows Form -sovellus tehtiin C# kielellä käyttäen sellaisia kirjastoja, jotka ovat käytössä myös Windows Phone 7.1 -ympäristössä. Jos ulkopuolisia kirjastoja ei käytetä, JSON-muunnokseen käytettävä kirjasto on System.Runtime.Serialization.Json.DataContractJsonSerializer (MSDN 2012a).

Myös .NET -ympäristössä siirrettävät luokat kopioitiin tuntikirjaussovelluksesta ja tietokanta-annotaatiot poistettiin. C# ja Java ovat kielinä niin läheisiä, että vain muutamia tyyppimuunnoksia tarvittiin. Javan kokoelma Set ja aikamääre Date voidaan C#:ssa korvata kokoelmalla HashSet ja aikamääreellä DateTime. Muunnettavat luokat ja kentät pitää annotaatiolla merkitä muunnettaviksi sarjamuotoon. Annotaatiolla määritetään tarkasti siirrettävän tiedon sisältö, ei niinkään tiedon tarkkaa tyyppiä. Kukin muunnettava luokka merkitään annotaatiolla [DataContract]. Kaikki muunnettavat kentät merkitään annotaatiolla [DataMember] ja kentille erikseen määrätään get- ja set-rajapinnat. Kentät, joita ei ole tarkoitus muuntaa JSON-muotoon, vaikka niilläkin on get- ja set-rajapinnat, merkitään annotaatiolla [IgnoreDataMember]. (MSDN 2012e.)

.NET -ympäristössä päivämäärien ja ajan (DateTime) JSON-muunnos kapseloi päiväyksen muotoon ”`\/Date(...)\/`”. Kellonaika ilmoitetaan UTC-aikana ja aikaerona. (Le Roy 2008.) Jackson JSON -muunnin käyttää ajalle unix-tyyppistä aikaa eli millisekunteja vuoden 1970 alusta eikä .NET-muoto ole sen kanssa yhteensopiva. Aika on mahdollista muuntaa .NET -muuntimen yhteydessä yhteensopivaan muotoon mutta muunnos on hieman kömpelö. .NET -ympäristö ei myöskään sujuvasti muunna puuttuvia (null) aikoja.

Sovelluksessa käytetty tapa aikojen siirtämiseen on luoda normaalin DateTime-tyypisen jäsenmuuttujan rinnalle Uint64?-tyyppinen apukenttä. Kysymysmerkki tyypin perässä tarkoittaa sitä, että muuttujan arvo voi olla null. Normaali aikakenttä, jota sovellus sisäisesti käyttää, merkitään jätettäväksi JSON-muunnoksen ulkopuolelle (listaus 18, rivit 1–2). Vastaavasti apukenttä merkitään JSON-muunnokseen aikakentän nimellä (rivit 3–4). Apukentän set- ja get-metodit (rivit 6–17) muuntavat ajan ja päivämäärän sisäisen DateTime-tyypin ja unix-ajan välillä (rivit 9 ja 15). Muunnoksen yhteydessä pitää ottaa huomioon aikaero UTC-ajan ja paikallisen ajan välillä.

```
001    [IgnoreDataMember]
002    public DateTime startdate { get; set; }
003    [DataMember(Name="startdate")]
004    private UInt64? _startdate
005    {
006           get
007           {
008                  if (this.startdate.CompareTo(unixEpochEET) > 0)
009                         return (UInt64)(this.startdate -
                                  unixEpochEET).TotalMilliseconds;
010                  else return null;
011           }
012           set
013           {
014                  this.startdate =
015                         new DateTime(unixEpochEET.AddMilliseconds(
                                  Convert.ToUInt64(value)).Ticks,
                                  DateTimeKind.Unspecified);
016           }
017    }
```

LISTAUS 18: .NET aikamuunnos DateTime- ja Unix-ajan välillä

Jotta kirjausten tallentaminen asettaisi oikean ajan aikavyöhykkeestä sekä kesä- ja talviajasta riippumatta, pitäisi käyttäjän asettaa sovellukseen käytettävä aikaero UTC-aikaan nähden. Tämäkin tapa muuntaa palvelimelta luettavien kirjausten ajan väärin, jos kirjaus on tehty talviajassa ja lukeminen kesäajassa tai päinvastoin. Lopullinen ratkaisu aikaongelmaan lienee ajan tallentaminen .NET -ympäristössä jossain muussa kuin DateTime-muodossa. Unix-aika ja manuaalinen muunnos UTC-aikaan voi olla yksi vaihtoehto. Toinen vaihtoehto voi olla ajan sisäinen tallentaminen vaikkapa merkkijonomuodossa päiväyksenä, tunteina ja minuutteina.

4.2 Käyttäjän tunnistaminen

Käyttäjän tunnistamisen ja oikeuksien tarkistuksen eli autentikoinnin vaihtoehtoja olivat Basic-autentikointi, Digest-autentikointi ja Spring Securityn käyttäminen. Yksinkertaisin vaihtoehto olisi ollut Basic-autentikointi (Franks ym. 1999), jossa käyttäjän tunnus ja salasana lähetetään joka kerta palvelimelle Base-64 muodossa mutta täysin salaamatta. Tästä syystä Basic-autentikointi on melko turvaton, ellei yhteytenä käytetä suojattua SSL-yhteyttä (Franks ym. 1999, luku 4; Wikipedia 2012a; Apache

2012a). Digest-autentikoinnissa käyttäjän tunnuksesta ja salasanasta sekä palvelimen antamista tunnisteista lasketaan tiiviste (eräänlainen tarkistussumma), jonka perusteella palvelin tunnistaa käyttäjän. Spring Security olisi myös voinut olla käyttökelpoinen vaihtoehto. Tällöin mobiilisovelluksen olisi pitänyt pitää huoli palvelimen antamasta evästeestä ja mahdollisesti käyttää Spring Securityn login-sivua. Mikään näistä menetelmistä ei salaa verkon yli lähetettävää tietoa. Jos salaukseen on tarvetta, pitää käyttää SSL-yhteyttä tai varsinaista salausta siirrettävän tiedon mukaisesti.

Tuntikirjaussovelluksessa mobiilirajapinta käyttää Digest-autentikoinnin muunnelmaa, jossa käytetään vahvempaa algoritmia tarkistussumman laskentaan sekä tunnisteen ikää rajoittavaa aikaleimaa.

Tavallisessa Digest-autentikoinnissa lasketaan ensimmäinen tiiviste käyttäjätunnuksesta, palvelimen tunnisteesta ja käyttäjän salasanasta. Toinen tiiviste lasketaan http-metodin tyypistä (GET, POST, PUT ja DELETE) ja haetun resurssin url-polusta. Lopullinen vastaus on näistä kahdesta tiivisteestä ja palvelimen antamasta avaimesta (nonce) koostuva tiiviste. Tarkistussumman laskenta on esitetty kuviossa 7 ja kenttien kuvaus taulukossa 3.

Kullakin yhteysyrityksellä palvelin vertaa palvelimen tunnistetta omaan tunnisteeseensa sekä http-metodin tyyppiä ja resurssin url-polkua asiakassovelluksen antamaan tyyppiin ja polkuun. Lisäksi käyttäjä yritetään tunnistaa käyttäjätunnuksen perusteella. Jos käyttäjä on tunnettu, haetaan käyttäjän salasana. Lopuksi palvelin laskee kaikista näistä tiedoista sekä yhteyden *nonce*-avaimesta oman tiivisteensä. Jos kaikki vertailut sekä tiiviste ovat yhtäpitävät, yhteys on todettu kelvolliseksi ja siihen vastataan normaalisti. Muussa tapauksessa palvelin vastaa HTTP-virheellä 401 (unauthorized) ja uudella nonce-avaimella sekä palvelimen tunnisteella. Virheen saatuaan mobiilisovelluksen tulee tallentaa palvelimelta saamansa kentät ja laskea uusi tiiviste uudelleen lähetettävälle pyynnölle.

```
HA1 = MD5(A1) = MD5( username : realm : password )

HA2 = MD5(A2) = MD5( method : digestURI )

Response = MD5( HA1 : nonce : HA2 )
```

KUVIO 7. RFC 2096 digest

Normaalin digest-autentikoinnin MD5 tiivisteen sijasta tässä mobiilirajapinnassa tarkistussumma lasketaan Hash256-algoritmilla ja laskentaan on lisätty palvelimen aikaleima, jonka kuluttua umpeen palvelin laskee uuden nonce-arvon ja pyytää käyttäjää uusimaan tunnistautumisen. Näiden lisäksi palvelin luo nonce-arvon käyttäjän nimestä ja senhetkisestä ajasta, eikä laskennassa käytetä salasanaa, vaan sen tiivistettä (tietokannassa salasana on vain tiivisteenä). Salasanan tiivistettä käytetään laskennassa, mutta sitä ei missään vaiheessa siirretä verkon yli, joten salasanan tiivisteen on oltava molempien osapuolien tiedossa. Salasanan selvittäminen tiivisteestä ja muista kentistä on hyvin vaikeaa. Tarkistussumman laskenta on esitetty kuviossa 8 ja kenttien kuvaus taulukossa 3.

```
Response = SHA256( username : realm : passwordhash :
                   timestamp : method : digestURI : nonce )
```

KUVIO 8. Tuntikirjaussovelluksen mobiilirajapinnan digest

TAULUKKO 3. Tarkistussumman laskenta mobiilirajapinnassa

Kenttä	Kuvaus
`username`	Käyttäjän tunnus, client → server
`realm`	Palvelimen tunniste, palvelin antaa ja mobiilisovellus palauttaa
`password`	Selväkielinen salasana
`passwordhash`	Salasanan SHA256-tiiviste (SHA256(`password`))
`timestamp`	Palvelimen antama aikaleima jonka perusteella vanha tunniste (`nonce`) uusitaan, mobiilisovellus palauttaa
`method`	Http-tyyppi (GET, POST, PUT, DELETE), mobiilisovellus lähettää
`digestUri`	Halutun resurssin polku, mobiilisovellus lähettää
`nonce`	Palvelimen antama kertakoodi, joka on tiiviste käyttäjän tunnuksesta ja sen hetkisestä ajasta, mobiilisovellus palauttaa

Sovelluksen mobiilirajapinnassa kaikki palvelimen antamat kentät lähetetään takaisin seuraavan pyynnön mukana. Tällä tavalla palvelimen ei tarvitse tallentaa kenttiä käyttäjän tietoihin. Haittana on jokaisessa pyynnössä mukana kulkevien kenttien määrä. Palvelin hyväksyy tarkistussumman sekä Base64- että heksadesimaalimuodossa.

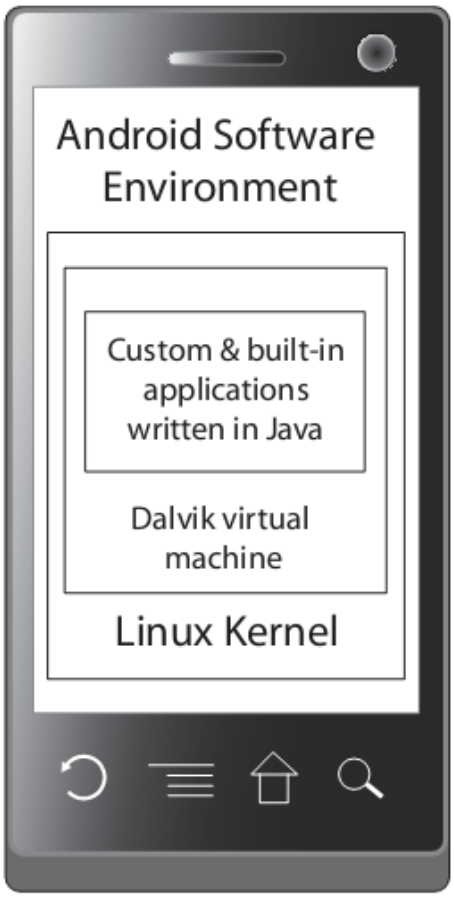
Windows Phone 7.1 -alustalla tiivisteet on laskettava käyttäen System.Security.Cryptography.SHA256Managed-luokkaa (MSDN 2012d). Internetin keskustelupalstoilla

toisinaan (esimerkiksi Judson 2011) esitetty System.Security.Cryptography.HMAC-SHA256 (MSDN 2012b) on käyttökelpoinen, jos tiiviste satunnaistetaan (suolataan) erillisellä avaimella.

Aikaleiman ansiosta mobiilitunnistus vanhenee samalla tavalla kuin selaimessa avoimeksi jätetty istunto ja vaatii myöhemmin uuden tunnistautumisen. Tunnistautuminen on mahdollista vain käyttäjälle, jolla on ROLE_USER käyttäjärooli. Mobiilirajapintaa voi käyttää vain tuntikirjausten tallentamiseen, muokkaamiseen ja lukemiseen, joten muilla käyttäjillä ei ole tarvetta käyttöoikeuteen.

5 ANDROID-SOVELLUS

Mobiilirajapinnan yhteydessä luvussa 4 sivuttiin syitä erilliselle mobiilisovellukselle. Tässä luvussa kuvataan Android-alustelle toteutettua mobiilisovellusta. Android on älypuhelimille ja tablet-laitteille tarkoitettu Linux-pohjainen käyttöjärjestelmä. Android on (ainakin periaatteessa) avoimen lähdekoodin ohjelmisto. Android-laitteiden ominaisuuksiin kuuluu aina kosketusnäyttö, langaton verkkoyhteys (Wi-fi tai datayhteys) ja vain harvoin mekaaninen qwerty-näppäimistö. Android-sovellukset ohjelmoidaan tavallisimmin Java-kielellä ja suoritetaan Java-virtuaalikoneessa, jonka Android-toteutuksesta Google käyttää nimeä Dalvik. (Meier 2010.) Kuva 4 esittää Androidin ohjelmistorakenteen.



KUVA 4. Androidin ohjelmistorakenne.
(Ableson, Sen, King & Ortiz 2012)

Android-sovelluksia tai niiden osia voi ohjelmoida myös C/C++ -kielillä käyttäen Android NDK:ta (Native Development Kit). NDK:n käyttäminen ei automaattisesti nopeuta sovellusta, mutta tekee siitä väistämättä monimutkaisemman. NDK soveltuu parhai-

ten esimerkiksi signaalien käsittelyyn tai fysiikan mallintamiseen. (Ableson & Sen & King & Ortiz 2012, luku 19) Lienee todennäköistä, että monet pelit (esimerkiksi Angry Birds) käyttävät NDK:lla toteutettuja osia.

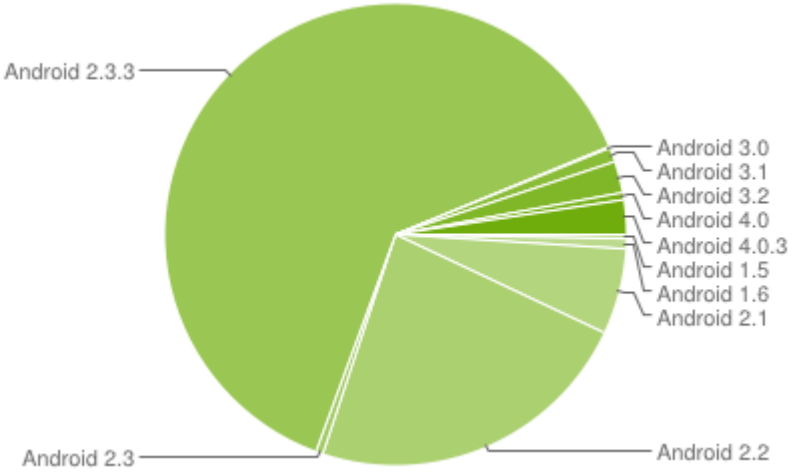
Android-sovellusten tärkein jakelukanava on vuonna 2008 perustettu Google Play -kauppa (aikaisemmin nimellä Android Market). Laitteen käyttäjä voi halutessaan sallia sovellusten asentamisen myös muista lähteistä, ja vaikkapa asentaa muistikortille ladatun sovelluksen.

5.1 Android-version valinta

Tuntikirjaussovellus on Android-alustalla suunnattu älypuhelimiin, tarkoituksena kattaa suuri osa käytössä olevista laitteista. Suurin osa tällä hetkellä käytössä olevista Android-laitteista (Google. 2012b. luettu 7.4.2012, tiedot 14 päivän ajalta 2.4.2012 asti) käyttää Android-versiota 2.3. Koska myös versio 2.2 on vielä käytössä melkein joka neljännessä laitteessa, oli perusteltua tehdä tuntikirjaussovellus toimimaan versiosta 2.2 alkaen. Tablet-laitteiden vähäinen määrä (käytännössä kaikki Android 3.x laitteet ja osa Android 4.x laitteista) mahdollistaa tablet-laitteille tarkoitetun käyttöliittymän suunnittelun jättämisen myöhempään ajankohtaan. Taulukossa 4 korostettuna ne Android-versiot, joissa Android-versiota luultavasti käytetään, ja joissa sovellusta tulisi testata.

TAULUKKO 4. Android-versioiden jakauma. (Google 2012b)

Alusta	Kutsumanimi	API	Osuus
Android 1.5	Cupcake	3	0,3 %
Android 1.6	Donut	4	0,7 %
Android 2.1	Eclair	7	6,0 %
Android 2.2	**Froyo**	**8**	**23,1 %**
Android 2.3 – 2.3.2	**Gingerbread**	**9**	**0,5 %**
Android 2.3.3 – 2.3.7		**10**	**63,2 %**
Android 3.0	Honeycomb	11	0,1 %
Android 3.1		12	1,0 %
Android 3.2		13	2,2 %
Android 4.0 – 4.0.2	**Ice Cream Sandwitch**	**14**	**0,5 %**
Android 4.0.3		**15**	**2,4 %**



KUVIO 9. Android-versioiden jakauma. (Google 2012b)

Koska kukin Android-versio on taaksepäin yhteensopiva (Ableson ym. 2012, 602), pitäisi sovelluksen toimia myös uusimmissa älypuhelimissa Android 4.x versiossa. Sovellusta on testattu taulukossa 5 luetelluilla kokoonpanoilla.

TAULUKKO 5. Testatut Android-ympäristöt

Android-versio	Näytön koko	Ympäristö
2.2	320 x 240	Emulaattori
2.2	320 x 480	Emulaattori
2.2	480 x 800	Emulaattori
2.3.6	320 x 480	Laite
4.0	480 x 800	Emulaattori

5.2 Yleistä Android-sovelluksesta

Android-sovellukseen voi kuulua käyttöliittymiä (Activity), taustapalveluja (Service), vastaanottimia (Broadcast Receiver) ja sisällön tarjoajia (Content Provider). Sovelluksessa ei useinkaan ole kaikkia edellä mainittuja osia. (Smith & Friesen 2011.) Tuntikirjaussovelluksessa ei esimerkiksi ole yhtään taustapalvelua, mutta sellainen voidaan jossain vaiheessa lisätä päivittämään tietoja sovelluksen ja verkkopalvelimen välillä silloinkin, kun sovellus on suljettuna.

Tuntikirjaussovelluksessa on useita käyttöliittymiä eri käyttötarkoituksia varten (käyttöliittymistä tarkemmin luvussa 5.4). Kuhunkin käyttöliittymään kuuluu android.app.Activity-luokasta laajennettu aktiviteettiluokka. Aktiviteettiluokka määrittää

näkymän (View) ja toteuttaa käyttöliittymän toiminnallisuuden. Näkymän osat (painikkeet, tekstikentät jne.) luodaan ja sijoitetaan näytölle ajon aikana joko ohjelmallisesti tai lataamalla etukäteen luotu näkymän määrittävä näkymätiedosto. Yhdellä aktiviteetilla voi olla useita näkymätiedostoja erikokoisia näyttöjä ja näytön eri asentoja varten.

5.2.1 Android-sovelluksen elämänkaari

Android-sovelluksen elämänkaaresta kerrataan lyhyesti taulukossa 6 aktiviteetin (Activity) toiminnot eri tilanteissa.

TAULUKKO 6. Android-aktiviteetin siirtymämetodit

Metodi	Milloin	Käyttö
`onCreate`	Aktiviteetti luodaan	Luo aktiviteetti ja kentät. Avaa tietokantayhteys ja lue tarvittavat tiedot. Palauta aktiviteetin pysyvät tiedot preferensseistä. Palauta aktiviteetin tila (esim. tekstikenttien teksti, fokus)
`onRestart`	Aktiviteetti palaa näytölle	
`onStart`	Aktiviteetti tulee näytölle	
`onResume`	Aktiviteetti saa fokuksen	
`onPause`	Aktiviteetti menettää fokuksen	Tallenna aktiviteetin pysyvät tiedot preferensseihin
`onStop`	Aktiviteetti poistetaan näytöltä	
`onDestroy`	Aktiviteetti poistetaan muistista	Sulkee tietokantayhteyden
`onSaveInstanceState`	Aktiviteetti voi poistua muistista. Ei kutsuta, jos käyttäjä poistuu aktiviteetista	Tallenna aktiviteetin tila (esim. tekstikenttien teksti, fokus)

Tietojen tallennuksessa tulee ottaa huomioon tallennettavien tietojen laatu. Pysyvät tiedot, joiden tulee pysyä muistissa sovelluksen tai aktiviteetin käynnistymisestä toiseen, tallennetaan onPause-metodissa pysyvään muistiin, preferensseihin. Tuntikirjaussovelluksessa tällaisia tietoja ovat esimerkiksi ”muista salasana” merkintä sekä salasana. Muuttuvat tiedot, kuten syöttökenttien arvo, tallennetaan onSaveInstanceState-metodissa bundleen. Muuttuvat tiedot tallennetaan vain silloin, kun käyttäjä ei sulje aktiviteettia painamalla takaisin-nappia. Muuttuvien tietojen tallennus palauttaa

käyttöliittymän siinä tilassa jossa se oli, jos välillä käytetään jotain toista sovellusta, tai laite käännetään pystyasennosta vaaka-asentoon tai päinvastoin.

Sekä pysyvät että muuttuvat tiedot (jos niitä on tallennettu) palautetaan onCreate-metodissa. Jos aktiviteetti lataa suuren määrän tietoja onCreate-metodin yhteydessä, voi asennon vaihdoksen ja siihen liittyvän näkymän uudelleenlataamisen hoitaa näkymän tuhoamisen ja uudelleen luomisen sijasta onConfigurationChange-metodissa.

5.2.2 Käyttöliittymän ja ohjelman yhteydestä

Android-sovellusten käyttöliittymä määritetään tavallisesti xml-tiedostoon. Ohjelmallista käyttöliittymän luontia käytetään yleensä dialogien luomisen yhteydessä, mutta myös koko käyttöliittymän voi luoda ohjelmallisesti onCreate-metodissa. Käyttöliittymän graafinen luominen tapahtuu Eclipsessä, jossa xml-tiedostoa voi muokata graafisen ulkoasun lisäksi myös tekstinä. Käyttöliittymän kullekin elementille (painonappi, tekstikenttä jne.) annetaan kuvaava tunniste. Näistä tunnisteista käännetään automaattisesti R-luokka, joka pitää sisällään kaikki näkymät, näkymien elementit ja muut resurssit. Jos käyttöliittymän elementtejä halutaan käyttää sovelluksessa, pitää kukin elementti erikseen hakea samantyyppiseksi objektiksi. Jos käyttökohteita on useita, kannattaa elementti tallentaa jäsenmuuttujaksi. Tämä prosessi on virhealtis, eikä virheitä yleensä huomaa ennen kuin näkymää testattaessa. NetBeansin ja Visual Studion kaltaista automaattista jäsenmuuttujien luomista käyttöliittymän elementeistä ei siis kannata odottaa.

Edellä mainittua elementtien hakemista R-kuokasta voi joissakin tapauksissa välttää. Jos käyttöliittymän elementtiin liittyy kosketuksesta kutsuttava metodi, sen voi liittää elementtiin *onClick*-ominaisuutena. OnClick-ominaisuudessa määritetään kutsuttava metodi, joka on oltava samannimisenä näkymän toteutuksessa. (Collins, Galpin & Käppler 2012, 123.) Jos käyttöliittymän elementti ei luonnostaan ole kosketettava (esimerkiksi tekstikenttä), voi elementin määrittää sellaiseksi *Clickable*-ominaisuudella. Näitä ominaisuuksia kuvataan listauksessa 19.

```
hourentry.xml:
001    <TextView
002           android:id="@+id/textView1"
003           android:clickable="true"
004           android:onClick="onEntrySetStart"
005           android:text="@string/hours_starttime"
006    />

HourEntryActivity.java:
001    public void onEntrySetStart (View v)
002    {
...
004    }
```

LISTAUS 19. Käyttöliittymän elementtiin liitetty metodi

5.3 Internet-yhteys

Android-sovellus käyttää internet-yhteyttä http-rajapinnan kautta. Sovelluksen ei tarvitse erikseen tietää minkä tyyppinen käytettävä yhteys on (Wi-fi, 3G, GPRS). Tuntikirjaussovellus käyttää internet-yhteyttä hakiessaan tietoja tuntikirjaussovelluksen verkkopalvelimelta tai lähettäessään uusia tai muuttuneita tuntikirjauksia. Varsinainen tieto siirtyy JSON-muodossa, ja yhteyspyyntöihin liittyy käyttäjän tunnistamiseen tarvittavat tiedot.

5.3.1 HttpUrlConnection

Internet-yhteyden toteuttaminen aloitettiin kopioimalla Java-testisovelluksen lähdekoodi soveltuvin osin Android-sovellukseen. Java-testisovelluksessa käytettiin ilman ongelmia samanlaista internetyhteyden muodostamistapaa java.net.HttpUrlConnection-luokalla kuin mobiiliarkkitehtuurit-kurssin harjoitustehtävissä. Yhteyden luominen ja käyttö toteutettiin erillisessä säikeessä käyttämällä AsyncTask-luokasta periytettyä sisäluokkaa. Sisäluokassa on se etu, että ulomman luokan muuttujat ja kentät ovat suoraan sisäluokan metodien käytettävissä.

HttpUrlConnection toi kuitenkin mukanaan vain ongelmia. Ensimmäinen vastoinkäyminen tuli palvelimen palauttaessa http-virheen 401 (unauthorized), jota käytetään silloin, kun käyttäjää ei voida tunnistaa ja palvelin palauttaa käyttäjälle uuden nonce-avaimen uutta tunnistautumisyritystä varten. HttpUrlConnection on automatisoitu vir-

hettä 401 silmälläpitäen siten, että se yrittää itsenäisesti ratkaista virheen Basic-autentikoinnilla, jos http-yhteyteen on liitetty autentikoija ja oikeat tunnukset. (Apache 2012b, ProcessResponseHeaders(), case HTTP_UNAUTHORIZED) Koska palvelin ei kuitenkaan käytä normaalia tunnistusta eikä palauta basic-autentikoinnin mukaisia kenttiä, tunnistus epäonnistuu. Keinoa poistaa automaattinen autentikointi ei löytynyt, joten ratkaisuksi kokeiltiin poistaa http-virhe 401 palvelimen päästä.

Seuraava este oli parametrien välitys palvelimelle. Parametrien välittäminen yhteyspyynnön otsikoissa on täysin tavallista, mutta Android 2.2 versiossa HttpUrlConnection on rikki, eivätkä parametrit välity palvelimelle oikein (Collins ym. 2012, 302). Tätä yritettiin kiertää sillä, että parametrit siirrettiin uri-osoitteessa (http://localhost:8080/TuntiMaKu/mobileapi/projects*?username=sussu@firma&nonce=ad6202dbeb027e7eefc571725ad0594dade31332dfa576047673507f*...). Yhtään kirjausta ei kuitenkaan saatu lähetettyä palvelimelle onnistuneesti. Lopulta tultiin siihen tulokseen, että virallisen Sun/Oracle-Javan ja Androidissa käytetyn Apachen HttpUrlConnection-toteutukset eivät vastaa toiminnallisuudeltaan toisiaan, ja päädyttiin kokeilemaan HttpClient-luokan toimintaa.

5.3.2 HttpClient

org.apache.http.client.HttpClient-luokasta käytettiin valmiiksi Android-käyttöön määritettyä versiota android.net.http.AndroidHttpClient. HttpClient suorittaa http-pyynnön (HttpGet, HttpPost, HttpPut tai HttpDelete) ja palauttaa palvelimen vastauksen HttpResponse-luokan oliossa. Tuntikirjaussovelluksessa internetyhteyttä varten luotiin kaksi apuluokkaa, joista toinen, HttpParamsHelper, pitää sisällään kaikki http-yhteyden parametrit (palvelun uri-osoite, tunnistamiseen tarvittavat kentät, tarkistussumman laskenta sekä lähetettävä data). Toinen apuluokka, HttpClientHelper, luo ja suorittaa varsinaisen http-pyynnön.

Listaus 20 esittää kuinka HttpClientHelper-luokkaa käytetään sovelluksessa. Http-pyynnön parametrit päivitetään ennen pyynnön suorittamista (rivit 2 ja 4). Tässä esimerkissä listanäkymässä haetaan http get-metodia käyttäen tuntikirjauksia palvelimelta, ja pyyntöön liitetään kyselynä ensimmäinen haettava kirjaus ja suurin palautettavien kirjausten lukumäärä (rivi 6, *rpar*). Jos toiminnaksi on valittu tuntikirjauksen poistaminen, http delete-metodille ei anneta kyselyä (rivi 8, *null*). Apuluokan metodi doString palauttaa palvelimelta tulleen vastauksen tekstimuodossa.

```
      HttpClientHelper hc;
...
001   if (ch == null)
002         ch = new HttpClientHelper(parhelper);
003   else
004         ch.setParams(parhelper);
005   if (parhelper.getMethod().equals(HttpParamsHelper.GET_method)) {
006         httpResponse = ch.doString(rpar);
007   } else {
008         httpResponse = ch.doString(null);
009   }
```

LISTAUS 20. Http-pyyntö sovelluksessa

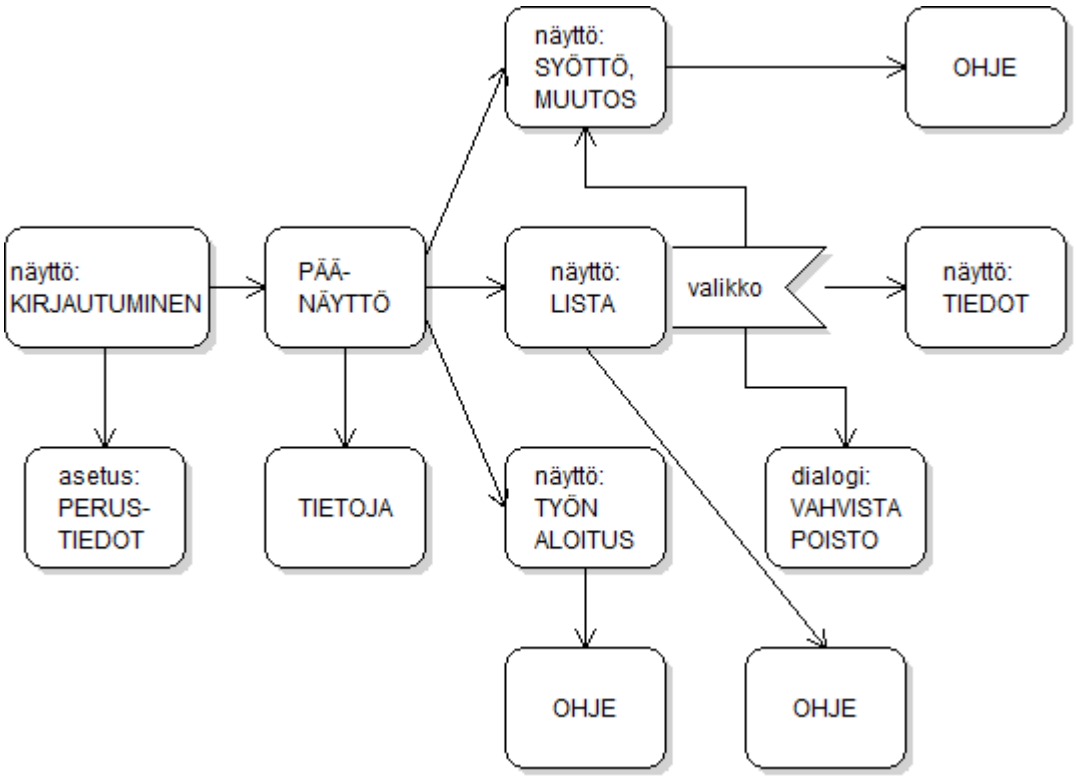
HttpClientHelper.doString-metodi on kuvattu listauksessa 21. Metodi doRequest (rivi 3) muodostaa annetuista parametreista ja kyselystä oikeanlaisen http-pyynnön, liittää siihen mahdollisen kyselyn ja suorittaa pyynnön. Jos http-metodina on post (tallenna uusi kirjaus) tai put (muuta kirjausta), kirjaus sijoitetaan ByteArrayEntity-olioon, joka asetetaan http-pyyntöön. Samalla pyynnölle annetaan parametri ”Content-Type” jonka arvoksi asetetaan "*application/json;charset=UTF-8*". doRequest-metodi palauttaa palvelimelta saadun HttpResponse-tyyppisen olion. Vastauksesta tarkistetaan onko tunnistautuminen onnistunut. Jos palvelin on palauttanut otsikoissa error-kentän jonka sisältö on ”AUTH”, kopioidaan palvelimen antama nonce-avain ja muut tunnistautumisessa tarvittavat kentät parametreihin ja suoritetaan pyyntö uudelleen. Saadusta vastauksesta luetaan getString-metodissa palvelimen palauttama tieto.

```
001   public String doString(Map<String,String> rparams) {
002         String r = null;
003         HttpResponse hr = doRequest(rparams);
005         Header errorheader = hr.getFirstHeader(
                  HttpParamsHelper.errorToken);
006         if (errorheader != null) {
007               String error = errorheader.getValue();
008               if (error != null && error.contains("AUTH")) {
009                     params.setNonce(hr.getFirstHeader(
                              HttpParamsHelper.nonceToken).getValue());
010                     params.setRealm(hr.getFirstHeader(
                              HttpParamsHelper.realmToken).getValue());
011                     params.setTimestamp(hr.getFirstHeader(
                              HttpParamsHelper.timeToken).getValue());
012                     hr = doRequest(rparams);
013               }
014         }
015         r = getString(hr);
016         return r;
017   }
```

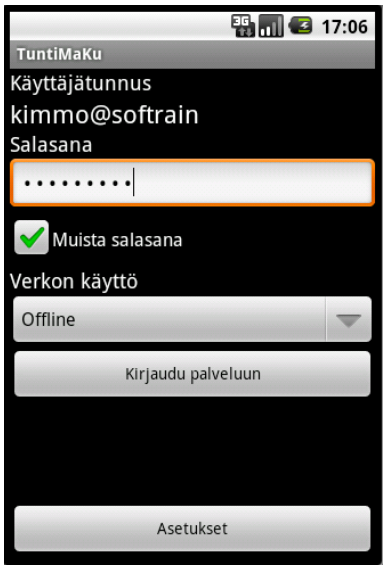
LISTAUS 21. Http-kutsu ja tunnistautuminen

5.4 Toiminnot ja käyttöliittymä

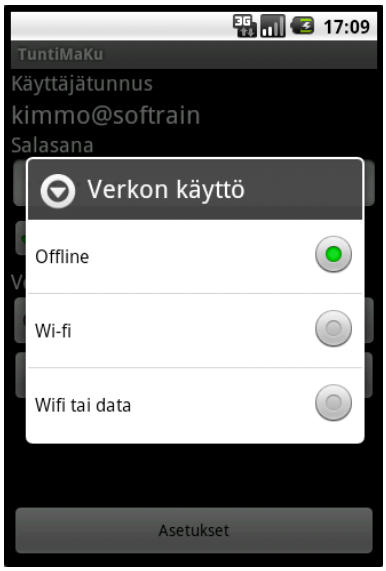
Käyttöliittymässä on käytetty dashboard-mallia (Fulcher, Nesladek, Palmer & Robertson 2010). Tässä mallissa sovelluksessa on etusivu, josta näkee helposti mahdolliset ilmoitukset sekä sovelluksen toiminnat. Päänäkymä (dashboard) on kuvion 10 näyttökartan päänäyttö. Näyttökarttaan on merkitty myös muut näkymät ja eteneminen niiden välillä. Tärkeimmät näytöt on esitetty kuvissa 5–14.



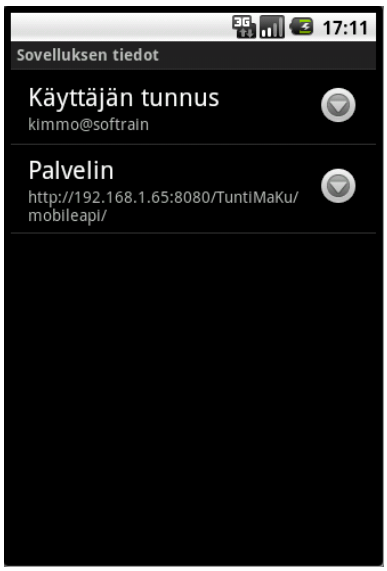
KUVIO 10. Sovelluksen näytöt



KUVA 5. Kirjautumisnäkymä



KUVA 6. Verkon käytön valinta

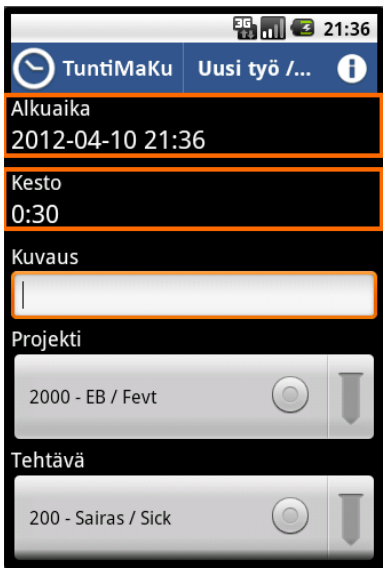


KUVA 7. Perusasetukset

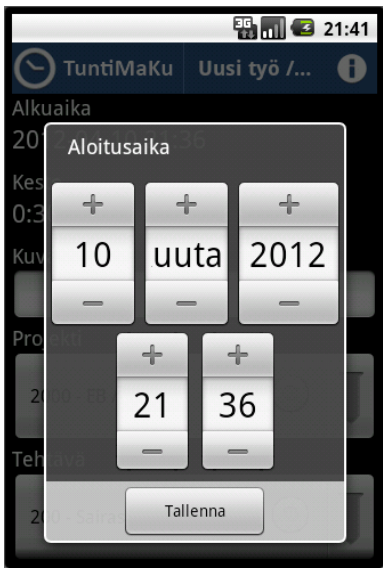
KUVA 8. Päänäkymä "dashboard"

KUVA 9. Listanäkymä

KUVA 10. Kirjausten pikavalikko

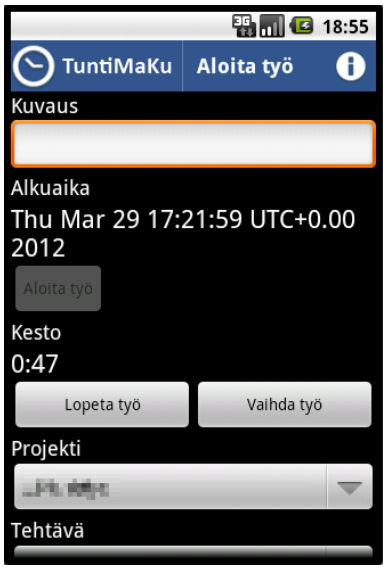


KUVA 11. Uusi kirjaus, kenttien kosketus-alueet



KUVA 12. Uusi kirjaus, aloitusajan syöttäminen

KUVA 13. Uusi kirjaus, keston asetus



KUVA 14. Reaaliaikainen syöttö

5.4.1 Kirjautuminen

Kirjautumisnäkymä esitetään kuvassa 5. Kirjautumisnäkymässä käyttäjä syöttää oman tuntikirjauspalvelun salasanansa ja mahdollisesti antaa sovelluksen tallentaa käyttäjän salasanan sovelluksen asetuksiin. Käyttäjätunnus ja tuntikirjauspalvelun osoite verkkopalvelimella määritellään asetuksissa.

Sovelluksen käyttö kalliiden datayhteyksien päässä, esimerkiksi ulkomaan komennuksilla, otetaan huomioon yhteystyypin valinnalla. Vaihtoehtoina ovat yhteydetön käyttö, wi-fi-yhteys (WLAN) sekä wifi- ja datayhteyksien (2G ja 3G data) käyttö. Yhteydettömässä käytössä ei palvelimeen oteta yhteyttä, vaan kaikki toiminta perustuu aikaisemmin laitteen sisäiseen tietokantaan haettuihin tietoihin. Myös uudet kirjaukset tallennetaan laitteen tietokantaan.

Wi-fi- ja datayhteydellä kirjautumisen yhteydessä haetaan aina viimeisimmät käyttäjän ja projektien tiedot. Uudet kirjaukset lähetetään palvelimelle ja tallennetaan laitteen sisäiseen tietokantaan. Vanhoja kirjauksia tarkasteltaessa tiedot haetaan palvelimelta ja tallennetaan laitteen sisäiseen tietokantaan yhteydettömässä tilassa käytettäväksi. Pelkkä wi-fi-yhteys toimii WLAN-yhteydellä (esimerkiksi hotellin WLAN-verkossa) kuten edellinen, mutta wi-fi-yhteyden puuttuessa sovellus ei käytä datayhteyttä vaan toimitaan kuten yhteydettömässä tilassa laitteen sisäisen tietokannan varassa.

5.4.2 Asetukset

Asetusnäyttö on luotu suoraan sovellukseen määritellyistä asetuksista. Molemmat asetukset ovat tekstityyppisiä (EditTextPreference). Asetusnäkymä esittää asetuksien arvot, kun asetusnäkymän aktiviteetti toteuttaa OnSharedPreferenceChangeListener-rajapinnan (Meier 2009). Rajapinnan metodi OnSharedPreferenceChangeListener.OnSharedPreferenceChanged suoritetaan aina muutoksen jälkeen ja metodissa voidaan asetuksen yhteenvedoksi asettaa haluttu teksti, tässä asetuksen arvo.

Mikäli myöhemmässä vaiheessa nähdään tarvetta useamman käyttäjän lisäämiselle sovellukseen, voidaan käytettävä asetustiedosto vaihtaa ohjelmallisesti (Android 2012b). Useamman käyttäjän tapauksessa voidaan käyttää esimerkiksi tilin käsitettä, jotta käyttäjänimet, palvelimet ja salasanat sekä muut asetukset voidaan kohdentaa kullekin käyttäjälle.

Useamman käyttäjän sovellusta voidaan käyttää työvuoroissa (tehtailla, sairaanhoidossa tai päivähoidossa, joissa älypuhelin ei ole henkilökohtainen vaan tehtäväkohtainen) tai jos työntekijä kirjaa eri tehtäviä eri palvelimille (alihankkija, jolla useampi toimeksiantaja). Useamman käyttäjän sovelluksessa myös tietokannat on hyvä erottaa toisistaan tai ainakin varmentaa, että toisen käyttäjän tietoihin ja kirjauksiin ei ole pääsyä.

5.4.3 Päänäkymä

Päänäkymä on niin sanottu dashboard (Fulcher ym. 2010) jossa käyttäjälle voi esitellä sovelluksen toiminnat ja edellisen käytön jälkeiset tapahtumat. Tuntikirjaussovelluksessa keskitytään perusasioihin: tuntikirjausten tekemiseen ja tarkastelemiseen.

Tuntikirjaussovelluksessa dashboard-näkymä perustuu Bill Lahden artikkeliin (Lahti 2011) ja esimerkkiin. Kuvassa 8 käytettävät kuvakkeet eivät ole sovelluksen lopulliset kuvakkeet. Kukin kuvaketta piirretään kolmessa eri tilassa (vapaa, valittu, painettu), sekä kolmessa eri koossa laitteiden näyttöjen koon ja tarkkuuden mukaan.

Näkymän alaosassa näkyvät aktiivisten projektien lukumäärä ja käyttäjälle määrättyjen projektien määrä, sekä laitteen sisäiseen tietokantaan tallennettujen kirjausten määrä, joita ei ole lähetetty palvelimelle. Näkymän yläosassa oikealla puolella olevaa kuvaketta koskettamalla saadaan päänäkymässä esille sovelluksen tiedot (versio, tekijä, tekijänoikeudet) ja muissa näkymissä lyhyt käyttöohje. Yläosassa vasemmalla puolella olevasta kuvakkeesta voidaan muista näkymistä palata päänäkymään.

Päivitä-kuvakkeesta avautuu näkymä, jossa voidaan määrätä mitä tietoja palvelimen kanssa päivitetään. Vaihtoehtoina ovat tallentamattomien tuntikirjausten lähetys palvelimelle, valitun ajanjakson tuntikirjausten haku palvelimelta sekä käyttäjän ja projektien tietojen päivittäminen palvelimelta. Lisäksi vanhoja tuntikirjauksia voi poistaa laitteen muistista. Tässä näkymässä päivitys tapahtuu riippumatta kirjautumisnäkymässä valitusta verkon käytöstä, koska kyseessä ei ole automaattinen toiminta. Päänäkymästä avautuvat muut toiminnat käydään läpi seuraavissa luvuissa.

5.4.4 Listanäkymä

Listanäkymään haetaan näkymän avautuessa 10 ajallisesti viimeisintä kirjausta, tai kuluvan viikon kirjaukset, ja kirjauksia voi selata vanhempiin ja takaisin uudempiin 10 kirjausta tai viikko kerrallaan. Jos käyttäjän valitsema verkkoyhteys on käytössä, haetaan kirjauksia myös palvelimelta. Palvelimelta haetut kirjaukset tallennetaan laitteen sisäiseen tietokantaan. Uudet ja muutetut kirjaukset, joita ei ole lähetetty palvelimelle, näkyvät listalla korostettuina.

Listanäkymä on toteutettu kaksirivisellä näkymällä (Piancastelli 2011), jossa ensimmäisellä rivillä näytetään tuntikirjauksen kuvaus ja kesto, ja toisella rivillä projekti ja aloitusaika. Listan rivimäärä tulee käytetyltä adapterilta, joka hakee tiedot näkymään.

Listanäkymän kussakin elementissä on kaksi riviä, joista ylemmän tunniste on android.R.id.text1 ja alemman android.R.id.text2.

Listaa varten luodaan taulukko data (listaus 22, rivi 1), jonka kullekin riville tallennetaan kaksi nimi-arvo-paria (rivi 3). Nimi-arvo-pareista ensimmäisen nimi on date ja sen arvoksi asetetaan tuntikirjauksen projekti sekä alkuaika (rivi 4). Toisen nimi-arvo-parin nimi on title ja sen arvoksi asetetaan tuntikirjauksen kuvaus ja kesto (rivi 5). Adapterissa (rivi 8) asetetaan listan kaksirivinen muoto (android.R.layout.simple_list_item_2), näytettävät tiedot (data), sekä sidotaan toisiinsa elementit ja näytettävät arvot (”title” ↔ android.R.id.text1 ja ”date” ↔ android.R.id.text2). Lopuksi adapteri asetetaan listan tietolähteeksi (rivi 9).

```
001   data = new ArrayList<Map<String, String>>();
002   for (Workhours wh : LocalDataStorage.loadedHours) {
003         Map<String, String> kirjausRivi =
                  new HashMap<String, String>(2);
004         kirjausRivi.put("date", wh.getProject().getName() + ":"
                  + wh.getTask().getName() + " - "
                  + wh.getStartdate().toString());
005         kirjausRivi.put("title",  wh.getDescription() +
                  " (" + wh.getHours() + " h)");
006         data.add(rivi);
007   }
008   tuntiAdapteri = new SimpleAdapter(context, data,
            android.R.layout.simple_list_item_2,
            new String[] { "title", "date" },
            new int[] { android.R.id.text1, android.R.id.text2 });
009   tunnitListView.setAdapter(tuntiAdapteri);
```

LISTAUS 22. Kaksirivisen listan luominen

Listanäkymään on lisätty pikavalikko, joka tulee näkyviin, kun listanäkymän riviä painaa jonkin aikaa. Pikavalikko mahdollistaa yleisimmät toimenpiteet. Tuntikirjaussovelluksessa valikko sisältää valitun kirjauksen tarkemman tarkastelemisen, valitun kirjauksen muuttamisen ja poistamisen sekä uuden kirjauksen luomisen. Jos esimies on hyväksynyt kirjauksen, muuttaminen ja poistaminen eivät ole mahdollisia.

Pikavalikon muokkaaminen esitetään listauksessa 23 riveillä 7–9. Rivillä 7 tarkastetaan onko kirjaus hyväksytty, ja seuraavilla riveillä pikavalikon toinen (muuta) ja kolmas (poista) valinta poistetaan käytöstä.

```
001   public void onCreateContextMenu(ContextMenu menu, View v,
             ContextMenuInfo menuInfo) {
002          super.onCreateContextMenu(menu, v, menuInfo);
003          AdapterContextMenuInfo info =
                    (AdapterContextMenuInfo) menuInfo;
004          MenuInflater inflater = getMenuInflater();
005          inflater.inflate(R.menu.listmenu, menu);
006          Workhours w = LocalDataStorage.loadedHours
                         .get(info.position);
007          if (w.getApproved() != null && w.getApproved() == true) {
008                menu.getItem(1).setEnabled(false);
009                menu.getItem(2).setEnabled(false);
010          }
011   }
```

LISTAUS 23. Pikavalikon muokkaaminen

5.4.5 Kirjausten syöttäminen ja muuttaminen

Android-sovelluksessa kirjauksen syöttäminen ei anna mahdollisuutta antaa lopetusaikaa. Tällä tavalla syötettäviä kenttiä on yksi vähemmän. Lisäksi työn kesto on helpompi syöttää kuin lopetusaika. Sekä alkuaika että kesto koostuu kahdesta tekstikentästä. Molemmat kentät ovat käytön helpottamiseksi koko näytön levyisiä ja molempien kosketus avaa syöttöikkunan. Tekstikentän muuttamisesta napautettavaksi kerrottiin kappaleessa 5.2.2. Lisäksi aloitusaika ja kesto on erotettu toisistaan, ja muista kentistä, pienellä välillä. Kosketusalueet on havainnollistettu kuvassa 11.

Kirjauksen aloitusajan ja keston asettaminen tapahtuu erillisissä syöttöikkunoissa (Android 2012a). Aloitusaika asetetaan kahdella eri komponentilla, jotka on liitetty valintaikkunaan päällekkäin. Päivämäärän asettamiseen käytetään *DatePicker*-komponenttia ja kellonaika asetetaan *TimePicker*-komponentilla. Komponenttien päivämäärä ja aika asetetaan, kun valintaikkuna luodaan, ja vastaavasti luetaan komponenteista, jos käyttäjä haluaa tallentaa aloitusajan. Kuvassa 12 huomataan, että kuukauden pitkä nimi ei mahdu valintaruutuun, kun laitteen käyttömaaksi on valittu Suo-

mi. Päivämäärän asettamiseen on siis löydettävä tapa, jossa esitetään suomen kieliset kuukausien nimet lyhyinä versioina (tam, hel, maa jne.), tai haettava kokonaan toinen malli, kuten kalenterinäkymä tai Windows Phone -tyylinen päivämäärän valinta.

Työn kesto syötetään valintaikkunassa jossa on pelkkä TimePicker-komponentti. Komponentilla voi asettaa ajan välillä 0:00 ja 23:59. Käytännössä tällainen kesto lienee riittävä, pitempiä yhtäjaksoisia työjaksoja on vaikea kuvitella. Esimerkiksi pitkät matka-ajat saattaisivat tulla kyseeseen, mutta näissä haitta lienee pieni.

5.4.6 Kirjauksen syöttö reaaliajassa

Tuntikirjauksia voi syöttää myös sitä mukaa, kun tehtävät vaihtuvat. Tätä varten on erillinen näkymä, jossa aloitusaika asettuu automaattisesti ja kesto laskee kulunutta aikaa aloitusajasta nykyhetkeen. Aloitusaikaa voi halutessaan muuttaa samalla tavalla kuin tavallisen kirjauksen yhteydessä. Kun työ on tullut valmiiksi tai vaihdetaan eri tehtävään, syötetään tehdylle työlle kuvaus, valitaan oikea projekti ja tehtävä sekä valitaan joko *lopeta työ* tai *vaihda työ*. Molemmat tallentavat kirjauksen laitteen sisäiseen tietokantaan ja lähettävät sen palvelimelle, jos käyttäjän valitsema verkkoyhteys on saatavilla. Valinta *vaihda työ* alkaa laskea uuden tehtävän kestoa. Jos näkymästä poistutaan, aloitusaika asettuu nykyhetkeen ja keston laskeminen alkaa alusta.

6 YHTEENVETO JA POHDINTA

6.1 Verkkopalvelualustat

Kaikissa kokeilluissa verkkopalvelualustoissa oli paljon hyvää: asentaminen ja alkuun pääseminen on nopeata ja helppoa ja ohjeita on riittävästi. Kirjallisuutta oli saatavissa kaikkiin kokeiltuihin ympäristöihin riittävästi. Lähes kaikki kirjat käyvät läpi perusasiat, mutta vähänkin erikoisemmat aiheet saatetaan sivuuttaa muutamalla rivillä. Kirjoihin kannattaa mahdollisuuksien mukaan tutustua etukäteen, jotta löytää tarpeitaan vastaavan oppaan. Internetissä melkein kaikki kysymykset on jo kysytty ja melko varmasti niihin vastattu. Erilaiset johdatukset sekä Stackoverflow ovat parhaita lähteitä.

Spring Roon etuna on CRUD-näkymien päivitys automaattisesti muiden muutosten myötä, tosin näkymiin käsin tehdyt muutokset ovat samalla vaarassa. Automaattinen päivitys sopiikin lähinnä pääkäyttäjän näkymiin. Spring Roon jspx-merkinnät (tag) ovat hyvin tiiviitä ja aluksi hankalia ymmärtää. Ajan myötä Roon merkintöihin kuitenkin tottuu, vaikka joissakin kohdissa edetäänkin enemmän ohjelmoijan vaistolla kuin tiedolla.

Grails vastaa CRUD-näkymien luontiin ja päivitykseen ajon aikana luotavilla näkymillä, jotka ovat aina ajan tasalla. Näkymiin voi lisätä omia toimintoja sekä muuttaa oletustoimintoja. Groovy on kielenä erilainen kuin Java tai C#, ja sen syntaksin opettelemiseen menee aikaa. Esimerkkejä ja generoituja ohjelmia muutellessa pärjää varsin hyvin, kun seuraa ja toistaa valmiin ohjelman syntaksia. Grailsia kokeiltiin NetBeans-ohjelmointityökalun kanssa, mutta ympäristön pitäisi toimia myös Eclipse-ohjelmointityökalun kanssa.

ASP.NET MVC 3 on joukosta ehkä helpoimmin lähestyttävä, osittain hyvän Visual Studio 2010 -ohjelmointityökalun ansiosta. Myös Razor-merkintäkieli on Grailsin tapaan helppolukuista ja käyttäjän hallinta on lähes valmiiksi mukana. Toisaalta CRUD-näkymiä ei voi päivittää automaattisesti, vaan muutokset pitää tehdä käsin tai luoda näkymät kokonaisuudessaan. Myös eri kielien tuki ASP.NET MVC 3 -ympäristössä jäi kokeilussa toteamatta.

Kokeilluista verkkopalvelualustoista kaikki sopivat yrityksen käyttöön. Jatkossakin kannattanee keskittyä Java-pohjaisiin palvelualustoihin, kuten Springiin. Palvelualustan kanssa käyttöliittymänä voi olla Web MVC (mahdollisesti Ajax-toiminnoilla vahvis-

tettuna), Google Web Toolkit tai Vaadin. Grailsin sijaan voi harkita PHP-kieleen tutustumista jonkin palvelualustan (esimerkiksi Zend Framework) yhteydessä. Microsoft-ympäristöissä voidaan olettaa, että ASP.NET (MVC tai Web Forms) on ylläpitäjille toivotuin alusta verkkosovelluksille, ja tästä syystä ASP.NET voi olla yksi jatkossa opiskeltavista alustoista.

6.2 Tuntikirjaussovellus

Tuntikirjaussovellus saatiin opinnäytetyön puitteissa hyvään alkuun. Sovellusta voi käyttää yrityksen sisällä, kunhan sille löydetään sopiva palvelin. Opinnäytetyön alkuvaiheessa selvästi suunniteltua pitempään kestänyt suunnitteluvaihe sekä Spring Roo -ympäristön hankaluus viivästyttivät aikataulua. Sen jälkeen, kun ympäristö oli tullut tutuksi, sovelluksen toteutus eteni kuitenkin reippaasti. Suurin puute sovelluksessa on palkanlaskennan raportoinnin puute.

Jälkikäteen ei voi sanoa Springin ja Roon valinnan kehitysympäristöksi ja työkaluksi olleen väärä. Vaikka kokeiltavia verkkopalvelualustoja olisi testattu ennen tuntikirjaussovelluksen toteuttamisen aloitusta, olisi kokeiltavien alustojen joukko saattanut olla kuitenkin sama.

Tuntikirjaussovellus kaipaa yksinkertaisempaa käyttöliittymää ja graafisia komponentteja, joita Rosenberg ryhtyi hakemaan omassa opinnäytetyössään. Jos GWT-käyttöliittymä ei tuota haluttua lopputulosta, voitaneen kokeilla Vaadin-käyttöliittymää. Kolmantena vaihtoehtona on muuttaa käyttöliittymä Ajax-pohjaiseksi JavaScript-toteutuksen huolehtiessa graafisista komponenteista.

6.3 Android-sovellus

Aikataulussa jälkeen jääminen johti siihen, että Android-sovelluksen suunnittelu jäi varsin vähäiseksi eikä sovelluksen ulkoasuun ehditty panostaa. Myös sovelluksen toteutus kärsi HttpUrlConnection-pohjaisen tietoliikenneyhteyden ongelmista, ennen kuin se vaihdettiin AndroidHttpClient-pohjaiseksi.

Android-sovelluksen toteutus tuntuu hankalalta, koska työkalut (Eclipse ja Android Development Tools) eivät toiminnaltaan vastaa esimerkiksi MicroSoftin Visual Studiota. Käyttöliittymän komponentit eivät tule automaattisesti sovelluksen toteutukseen, vaan ne pitää erikseen hakea näkymästä luodusta R-luokasta. Nyttemmin myös iPhone-ohjelmointia lyhyesti kokeilleena Android-sovelluksen kehitys on vielä hieman

raa'an oloista ja työkalut toivottavasti kehittyvät. Android on alustana niin suosittu, että Googlen pitäisi helpottaa sovellusten toteutusta mahdollisimman paljon.

6.4 Lopuksi

Opinnäytetyön myötä olen työparini kanssa saanut hyödyllistä kokemusta verkkopalveluiden ohjelmoinnissa käytettävistä alustoista ja tekniikoista. Työn edistyessä vaikeuksia on tullut vastaan, mutta niistä on paljolti myös selvitty. Jos alustojen eri komponentit ovat olleet keskenään yhteensopivia, on loppu ollut lähes kokonaan mekaanista logiikan toteutusta tarvittavan toiminnallisuuden aikaansaamiseksi. Mukaan on mahtunut myös pieniä ihmeitä, kun jokin käytetty tekniikka tai rajapinta on toiminut juuri niin kuin sen pitääkin.

Aikataulut ovat pettäneet lähes koko ajan. Aikatauluja on yritetty säätää käytettävissä olevan ajan puitteissa. Toisaalta käytettyjen alustojen ja tekniikoiden tuntemattomuus ei olisi muutenkaan mahdollistanut tarkempaa arviointia.

Mitä opinnäytetyön aikana sitten opittiin? Yhdeksän hyvää ja kymmenen kaunista, yhdeksän hyvää ja kymmenen kirosanaa (Irina – Yhdeksän hyvää ja kymmenen kaunista. 2010) kuvannee työtä parhaiten. Helppokäyttöisten alustojen avulla sovellusten toteuttamisessa pääsee lupausten mukaisesti hyvin helposti alkuun. Siitä eteenpäin joutuukin käyttämään omaa päätään ja hankkia tietämystä miten milläkin ohjelmistoalustalla toteutuksia oikeasti tehdään.

Erilaisia suunnittelu- ja toteutusmalleja tulee aktiivisesti etsiä ja hakea omaan tietovarastoon, jotta verkko- tai Android-sovelluksia ei tarvitsisi suunnitella aivan alusta alkaen. Sovellusten suunnittelun ja toteutuksen mahdollisuuksista tätä opinnäytetyötä tehtäessä on raaputettu vasta vähän pintaa. Toiseksi päällimmäinen kuva opinnäytetyön prosessista on se, että opittavaa on vielä todella paljon.

Päällimmäinen kuva on, onneksi: minä opin ja osaan ;)

LÄHTEET

Ableson, W. F., Sen, R., King, C. & Ortiz, C. E. 2012. *Android in Action, Third Edition.* Shelter Island: Manning Publications Co.

Alex, B. & Taylor, L. *Spring Security Reference Documentation* [verkkodokumentti]. [viitattu 5.5.2012]. Saatavissa http://static.springsource.org/spring-security/site/docs/3.0.x/reference/springsecurity.pdf.

Android 2012a. *Dialogs* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://developer.android.com/guide/topics/ui/dialogs.html.

Android 2012b. *PreferenceManager* [verkkosivu]. [viitattu 7.5.2012]. http://developer.android.com/reference/android/preference/PreferenceManager.html#setSharedPreferencesName(java.lang.String).

Apache 2012a. *Authentication, Authorization, and Access Control* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://httpd.apache.org/docs/1.3/howto/auth.html#basiccaveat.

Apache 2012b. *HttpURLConnectionImpl.java* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://www.java2s.com/Open-Source/Android/android-core/platform-libcore/org/apache/harmony/luni/internal/net/www/protocol/http/HttpURLConnectionImpl.java.htm.

Ayrenj. 2008. *MVC Architecture Diagram* [verkkosivu]. [viitattu 14.5.2012]. Saatavissa http://forums.asp.net/post/2493422.aspx.

Beckwith, B. & Talbott, B. 2012. *Spring Security Core Plugin - Reference Documentation* [verkkodokumentti]. [viitattu 6.5.2012]. Saatavissa http://grails-plugins.github.com/grails-spring-security-core/docs/manual/guide/single.pdf.

Collins, C., Galpin, M. D. & Käppler M. 2012. *Android in Practice*. Shelter Island: Manning Publications Co.

Crockford, D. 2006. *The application/json Media Type for JavaScript Object Notation (JSON)* [verkkodokumentti]. [viitattu 6.5.2012]. Saatavissa http://tools.ietf.org/html/rfc4627.

Davis, S. & Rudolph, J. 2010. *Getting Started with Grails, Second Edition* [verkkodokumentti]. USA: C4Media [viitattu 25.5.2012]. Saatavissa http://www.infoq.com/minibooks/grails-getting-started.

de Goul, M. O. 2012. *Epoch & Unix Timestamp Conversion Tools* [verkkosivu]. [viitattu 6.5.2012]. Saatavissa http://www.epochconverter.com/.

Donald, K. 2010. *Ajax simplifications in Spring 3.0* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://blog.springsource.org/2010/01/25/ajax-simplifications-in-spring-3-0/.

Dykstra, T. 2011. *Creating an Entity Framework Data Model for an ASP.NET MVC Application* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application.

Ekkelenkamp, R. 2009. *The grails application generator* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://grag.sourceforge.net/index.html.

Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. & Stewart, L. 1999. *HTTP Authentication: Basic and Digest Access Authentication* [verkkodokumentti]. [viitattu 7.5.2012]. Saatavissa http://tools.ietf.org/html/rfc2617.

Freeman, A. & Sanderson, S. 2011. *Pro ASP.NET MVC 3 Framework, Third Edition*. New York: Apress.

Fulcher, R., Nesladek, C., Palmer, J. & Robertson, C. 2010. *Android UI Design Patterns* [verkkodokumentti]. [viitattu 7.5.2012]. Saatavissa http://dl.google.com/googleio/2010/android-android-ui-design-patterns.pdf.

Gabor, T. 2012. *Online JSON Viewer* [verkkosivu]. [viitattu 6.5.2012]. Saatavissa http://jsonviewer.stack.hu/.

Galloway, J., Haack, P., Wilson, P. & Allen, K. S. 2011. *Professional ASP.NET MVC 3*. Indianapolis: John Wiley & Sons, Inc.

Google 2012a. *Google-gson, A Java library to convert JSON to Java objects and vice-versa* [verkkosivu]. [viitattu 6.5.2012]. Saatavissa http://code.google.com/p/google-gson/.

Google 2012b. *(Android) Platform Versions* [verkkosivu]. [viitattu 7.4.2012]. Saatavissa http://developer.android.com/resources/dashboard/platform-versions.html.

Grails 2012. *Grails* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://grails.org/.

Groovy 2012. *Groovy - A dynamic language for the Java platform* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://groovy.codehaus.org/.

Gulati, S. 2012. *Introducing Spring Roo, Part 2: Developing an application with Spring Roo* [verkkosivu]. [viitattu 8.5.2012]. Saatavissa http://www.ibm.com/developerworks/library/os-springroo2/.

Hanselman, S. 2011. *Getting started with MVC3* [verkkodokumentti]. [viitattu 7.5.2012]. Saatavissa http://www.avanic.com.ar/downloads/docs/getting_started_with_mvc3_cs.pdf.

Jackson 2012a. *Jackson High-performance JSON processor* [verkkosivu]. [viitattu 6.5.2012]. Saatavissa http://jackson.codehaus.org/.

Jackson 2012b. *Class ObjectMapper* [verkkosivu]. [viitattu 6.5.2012]. Saatavissa http://jackson.codehaus.org/1.6.6/javadoc/org/codehaus/jackson/map/ObjectMapper.html).

Johnson, R. & Hoeller, J. 2004. *Expert One-on-One™ J2EE™ Development without EJB™*. Indianapolis: Wiley Publishing, Inc.

Johnson, R., Hoeller, J., Donald, K., Sampaleanu, C., Harrop, R., Arendsen, A., Risberg, T., Davison, D., Kopylenko, D., Pollack, M., Templier, T., Vervaet, E., Tung, P., Hale, B., Colyer, A., Lewis, J., Leau, C., Fisher, M., Brannen, S., Laddad, R., Poutsma, A., Beams, C., Abedrabbo, T., Clement, A., Syer, D., Gierke, O. 2010. *The Spring Framework - Reference Documentation 3.0* [verkkodokumentti]. [viitattu 14.5.2012]. Saatavissa http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html.

Judd, C. M., Nusairat, J. F. & Shingler, J. 2008. *Beginning Groovy and Grails: From Novice to Professional*. Berkeley: Apress.

Judson, S. 2011. *Windows phone encrypting data using sha256* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://stackoverflow.com/a/8051381.

Kaukonen, K. & Ronkainen, E. 2011. *Tuntikirjaus ohjelmiston määrittely*. SoftRain Blobs Oy. Ei julkaistu.

Labunskiy, E. 2011. *Add User Roles on Registration (Forms Authentication)* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://stackoverflow.com/a/7283089.

Lahti, B. 2011. *How To Build A Dashboard User Interface In Android* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://blahti.wordpress.com/2011/03/14/build-dashboard-ui-for-android/.

Le Roy, B. 2008. *Dates and JSON* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://weblogs.asp.net/bleroy/archive/2008/01/18/dates-and-json.aspx.

Ledbrook, P. 2010. *Simplified Spring Security with Grails* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://blog.springsource.org/2010/08/11/simplified-spring-security-with-grails/.

Lerman, J. 2011. *Building an MVC 3 App with Database First and Entity Framework 4.1* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://msdn.microsoft.com/en-us/data/gg685489.

Lippert, M. 2012. *STS and JDK 64 bit* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://forum.springsource.org/showthread.php?75999-STS-and-JDK-64-bit&p=400560#post400560.

Long, J. & Mayzak, S. 2011. *Getting Started with Roo*. Sebastopol: O'Reilly.

Mak, G. 2008. *Spring Recipes: A Problem-Solution Approach*. Berkeley: Apress.

Meier, R. 2009. *How do I display the current value of an Android Preference in the Preference summary?* [verkkosivu] [viitattu 7.5.2012]. Saatavissa http://stackoverflow.com/a/531927.

Meier, R. 2010. *Professional Android 2 Application Development*. Indianapolis: Wiley Publishing Inc.

Microsoft 2012a. *ASP.NET MVC 3* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://www.asp.net/mvc/mvc3.

Microsoft 2012b. *Microsoft DreamSpark* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa https://www.dreamspark.com/.

Microsoft 2012c. *Visual Web Developer 2010 Express* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-web-developer-express.

Mkyong. 2010. *Spring MVC password example* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://www.mkyong.com/spring-mvc/spring-mvc-password-example/.

Mohan, V. 2011. *UTF-8 convertion issues* [verkkosivu]. [viitattu 6.5.2012]. Saatavissa http://jackson-users.ning.com/forum/topics/utf-8-convertion-issues?xg_source=activity.

MSDN 2012a. *DataContractJsonSerializer Class* [verkkosivu]. [viitattu 6.5.2012]. Saatavissa http://msdn.microsoft.com/en-us/library/system.runtime.serialization.json.datacontractjsonserializer(v=vs.95).aspx.

MSDN 2012b. *HMACSHA256 Class* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://msdn.microsoft.com/en-us/library/system.security.cryptography.hmacsha256.aspx.

MSDN 2012c. *Roles class* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://msdn.microsoft.com/en-us/library/39e0w7ay.aspx.

MSDN 2012d. *SHA256Managed Class* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://msdn.microsoft.com/en-us/library/system.security.cryptography.sha256managed.aspx.

MSDN 2012e. *Using Data Contracts* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://msdn.microsoft.com/en-us/library/ms733127.aspx.

Mularien, P. 2010. *Spring Security 3*. Birmingham: Packt Publishing.

MySQL 2012. *Tutorial: MySQL Connector/Net ASP.NET Membership and Role Provider* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://dev.mysql.com/doc/refman/5.5/en/connector-net-tutorials-asp-roles.html.

Neels. 2011. *Having problems with Spring Security plugin* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://grails.1312388.n4.nabble.com/Having-problems-with-Spring-Security-plugin-tp4041437p4079483.html.

NetBeans 2012. *Introduction to the Grails Web Framework* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://netbeans.org/kb/docs/web/grails-quickstart.html.

Piancastelli, G. 2011. *Android: Adding ListView Sub Item Text* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://stackoverflow.com/a/7917516.

Rahien, A. 2011. *Refactoring toward frictionless & odorless code: The case for the view model* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://ayende.com/blog/4807/refactoring-toward-frictionless-odorless-code-the-case-for-the-view-model.

Raible, M. 2011. *Java Web Application Security - Part II: Spring Security Login Demo* [verkkosivu]. [viitattu 6.5.2012]. Saatavissa http://raibledesigns.com/rd/entry/java_web_application_security_part1.

Robert. 2010. *Using Spring-Security Database in Spring-Roo* [verkkosivu]. [viitattu 6.5.2012]. Saatavissa http://roosbertl.blogspot.com/2010/06/using-spring-security-database-in.html.

Rocher, G., Ledbrook, P., Palmer, M., Brown, J., Daley, L. & Beckwith, B. 2012. *The Grails Framework - Reference Documentation, Version 2.0.3* [verkkodokumentti]. [viitattu 5.5.2012]. Saatavissa http://grails.org/doc/latest/guide/single.pdf.

Rosenberg, L. 2012. *Tuntikirjaus-ohjelmisto. Tietokannan ja web-käyttöliittymän suunnittelu sekä mobiilisovellus windows phonelle*. Opinnäytetyö. Kuopio: Savonia AMK.

Rosenberg, L. & Toivanen, K. 2011. *Tuntikirjaus, toiminnallinen määrittely*. SoftRain Blobs Oy. Ei julkaistu.

Scala, G. S. 2012. *MVC 3 jQuery Validation/globalizing of number/decimal field* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://stackoverflow.com/a/10230198.

Shklar, L. & Rosen, R. 2003. *Web Application Architecture, Principles, protocols and practices* [verkkodokumentti]. John Wiley & Sons Ltd [viitattu 16.5.2012]. Saatavissa http://www.ce.uniroma2.it/courses/PRSI/WebApplication.pdf.

Slauma. 2012. *EF 4.1 - Code First - JSON Circular Reference Serialization Error* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://stackoverflow.com/a/5588725.

Smith, D. & Friesen, J. 2011. *Android Recipes: A Problem-Solution Approach*. New York: Apress.

Smith, G. & Ledbrook, P. 2009. *Grails in Action*. Greenwitch: Manning Publications Co.

SpringSource 2007. *The Spring Framework - Reference Documentation* [verkkodokumentti]. [viitattu 5.5.2012]. Saatavissa http://static.springsource.org/spring/docs/2.5.x/reference/web-integration.html.

SpringSource 2011. *Invalid error marker on load-scripts.tagx file in resultant clinic.roo project* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa https://issuetracker.springsource.com/browse/STS-1915.

SpringSource 2012. *Spring Projects - Spring for Android* [verkkosivu]. [viitattu 15.5.2012]. Saatavissa http://www.springsource.org/spring-android.

Strumpflohner, J. 2011. *Posting JSON Data to an ASP.NET MVC 3 Web Application* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://blog.js-development.com/2011/08/posting-json-data-to-aspnet-mvc-3-web.html.

Tahvonen, M. & Grönroos, M. 2012. *Vaadin TouchKit in Action* [verkkosivu]. (viitattu 15.5.2012). Saatavissa http://demo.vaadin.com/vornitologist/VAADIN/tutorial/touchkit-tutorial.html.

Virta, M. 2009. *Aspektipohjainen .NET-ohjelmistokehitys*. Opinnäytetyö. Helsinki: Metropolia Ammattikorkeakoulu. Saatavissa http://urn.fi/URN:NBN:fi:amk-200911235878.

VMware 2011. *Spring Roo - Reference Documentation (1.2.1.BUILD-SNAPSHOT)* [verkkodokumentti]. [viitattu 5.5.2012]. Saatavilla http://static.springsource.org/spring-roo/reference/pdf/spring-roo-docs.pdf.

Walls, G. 2011. *Spring in action, Third Edition*. Shelter Island: Manning Publications Co.

Wheeler, W. 2008. *Hashing and salting passwords with Spring Security 2* [verkkosivu]. [viitattu 6.5.2012]. Saatavissa http://springinpractice.com/2008/10/11/hashing-and-salting-passwords-with-spring-security-2/.

Wikipedia 2012a. *Basic access authentication* [verkkosivu]. [viitattu 7.5.2012]. Saatavissa http://en.wikipedia.org/w/index.php?title=Basic_access_authentication&oldid=487291976.

Wikipedia 2012b. *Comparison of web application frameworks* [verkkosivu]. [viitattu 5.5.2012]. Saatavissa http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks.

VERKKOPALVELUALUSTAT JOISTA TUTKITTAVAT ALUSTAT ON VALITTU

Seuraavissa taulukoissa tyhjä kenttä voi tarkoittaa joko ”ei” tai ”ei tietoa”. Taulukkojen lähde (Wikipedia. 2012b.).

Alusta	Kieli	Ajax	MVC	MVC push-pull	i18n ja l10n
ASP.NET MVC	C#, VB.net	X	X	Push	
Spring	Java	X	X	Push	X
Apache Struts	Java	X	X	Push-pull	X
Apache Wicket	Java	YUI, ExtJS ym.	Modulaarinen	Pull	X
JavaServer Faces	Java	X	X	Pull	X
Play	Java	X	X	Push-pull	X
Vaadin	Java	GWT		Pull	X
Google Web Toolkit	Java, JavaScript	X			X
Grails	Groovy	X	Active record -malli	Push	X

Ajax malli, jossa selaimessa suoritettava JavaScript-ohjelma hakee uutta tai korvaavaa tietoa, esimerkiksi XML-muotoisena, selaimen näyttämälleen sivulle. Haku tapahtuu asynkronisesti eikä aiheuta koko sivun lataamista palvelimelta.

MVC sovellusarkkitehtuuri, jossa malli (data), näkymä ja kontrolleri (toiminnat) on erotettu toisistaan.

Push-pull hakeeko näkymä tietoja mallista (pull) vai lähettääkö kontrolleri tietoja näkymälle (push).

i18n sovelluksen rakentaminen siten, että sen voi sovittaa eri kielille ja kansallisuuksille (i*nternationalizatio*n: 18 merkkiä alku-i:n ja loppu-n:n välissä).

l10n sovelluksen sovittaminen jollekin tietylle kielelle ja kansallisuudelle (l*ocalizatio*n: 10 merkkiä alku-l:n ja loppu-n:n välissä).

Alusta	ORM	Testaus	Tietokannan rakenteen päivitys	Suojaus
ASP.NET MVC	ORM-riippumaton	Yksikkötestit		ASP.NET Forms Authentication
Spring	Hibernate, iBatis ym.	Yksikkötestit, testiobjektit		Spring Security
Apache Struts	X	Yksikkötestit		
Apache Wicket	Laajennuksella	Yksikkötestit, testiobjektit (laajennus)		X
JavaServer Faces	Laajennuksella	JUnit		X
Play	JPA, Hibernate	JUnit, Selenium	X	Core Security
Vaadin	X	X		
Google Web Toolkit	JPA (RequestFactory)	Selenium, JUnit, jsUnit	Javan kautta	X
Grails	GORM, Hibernate	Yksikkö- ja integrointitestit	Laajennuksella	Spring Security, Apache Shiro

ORM objektin (tai objektitietokannan) ja relaatiotietokannan sovittaminen toisiinsa.

Testaus työkalut automaattiseen testaukseen.

Päivitys millä tavalla sovellus voi hallitusti muuttaa tietokannan rakennetta elinkaarensa aikana, mieluiten tietoja kadottamatta.

Suojaus käyttäjän tunnistaminen, oikeuksien hallinta ja erilaisilta verkkohyökkäyksiltä puolustautuminen.

Alusta	Mallit	Välimuisti	Tarkistus
ASP.NET MVC	mm. Razor	X	X
Spring	JSP, Commons Tiles, Velocity ym.	X	Commons validator
Apache Struts	X		X
Apache Wicket	X	X	X
JavaServer Faces	Facelets, JSP	X	Sisäinen, Bean Validation
Play	X	X	Palvelimella
Vaadin Google Web Toolkit	X		X Bean Validation
Grails	X	X	X

Mallit näkymissä käytettävä tekniikka tai merkintäkieli.

Välimuisti sovellusalusta voi tallentaa tekemiensä hakujen tuloksia välimuistiin josta ne ovat seuraavalla kerralla nopeammin saatavissa.

Tarkistus kenttien tarkistuskohteiden määrittely (esimerkiksi pituus vähintään) ja kenttien tarkistus sekä virheen palauttaminen, jos tarkistusta ei läpäistä.


SAVONIA